UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

TRABAJO FIN DE GRADO

Búsqueda de imágenes asociadas

a documentos multitemáticos

Felipe Jasmin Sales

Tutor: Simone Santini

JUNIO 2014

RESUMEN

La tarea de añadir imágenes a documentos de texto es algo que no sólo lo deja más interesante sino también capta la atención del lector, haciendo más probable que este lo comparta.

Encontrar la mejor imagen para representar un texto es una tarea fácil cuando tratamos un único tema o tópico, pero se hace más difícil a medida que se añaden nuevos temas al mismo documento. Las personas pueden estar horas buscando la imagen deseada, sobre todo en los últimos años donde el número de imágenes almacenadas en la web ha crecido considerablemente, llegando a niveles que harían imposible la comprobación de todas las opciones disponibles antes de tomar una decisión.

Con este fin, el objetivo de este proyecto es el desarrollo de un método que facilite esta tarea para escritores de distintos medios. En lugar de tener que realizar la búsqueda en toda la web con diferentes palabras-clave (*key words*) y parámetros hasta encontrar lo que se quiere, podemos buscar en un conjunto más específico donde ya se incluyen las mejores imágenes encontradas.

Existen diversos métodos de búsqueda de imágenes asociadas a texto, muchos de los cuales se basan en la utilización de las palabras-clave (*key words*) de todo el texto. Aquí buscamos un nuevo enfoque, de manera que primero se segmenta el texto para obtener los fragmentos que lo componen. Luego se obtienen individualmente las imágenes para cada segmento usando las palabras más importantes (key words) y, por último se aplica una medida de similitud para obtener las mejores.

Este documento incluye los detalles del método implementado y del programa desarrollado que lo utiliza, para encontrar el conjunto de imágenes que mejor represente el texto inicial.

PALABRAS CLAVE

Recuperación de información, Recuperación de imágenes, segmentación de texto, comparación de imágenes, consulta de imágenes mediante ejemplo, Bing.

SUMMARY

Adding Images to a text document is a task that not only increases reader's interest but also grab his attention, making it more likely to be shared.

Finding the right images to represent a text may be easy when we have only one theme or topic in mind, but difficulty increases when we are dealing with large documents that include texts with more than one main topic. A writer may spend a long time trying to find the right image, especially in recent years, due to the fact that the amount of image data stored in the web has grown considerably; reaching levels that would make it impossible to look through all images before making a decision.

With that in mind, the aim of this project is to develop a method to help writers of every kind to achieve this task. Instead of having to search the whole web with different key words and parameters until we find what we want, we can search a smaller group that includes the best images found.

There are many available methods for retrieving images, most of which use the key words from the entire text on its search. The method proposed here differs from those methods, in a way that first the original text is segmented in order to obtain a new text fragment for each topic. We can then find images for each segment separately, using the most important words (key words) found in each of them, and apply a similarity measure to get the best ones.

This document includes the description of this method and the details of the developed program that achieves this task, so that we would be able to find the best group of images that represent the original text.

KEY WORDS

Information Retrieval, Image Retrieval, Content-based image retrieval, text segmentation, image Comparison, Bing.

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ECUACIONES

GLOSARIO

- **JAVA:** Lenguaje de programación orientado a objetos, basado en clases.
- **C:** Lenguaje de programación utilizado en la creación de software.
- **Eclipse:** Entorno de desarrollo multiplataforma que permite la implementación de aplicaciones a través de diversas herramientas y plug-ins.
- **OpenCV:** Librería de procesamiento de imágenes disponible para varios lenguajes de programación.
- **Key Words:** Palabras clave encontradas en un texto.
- **Captions:** Conjunto de palabras que describen una imagen.
- **Stemming:** Proceso en el que se elimina el final de una palabra con el fin de encontrar una base común.
- **Stem:** Palabra base obtenida tras aplicar el *stemming*.
- **Stop Words:** Palabras que aparecen en un texto y no añaden información relevante.
- **API:** *Application Programming Interface* – Conjunto de métodos que especifican como ciertas aplicaciones pueden interactuar con otras.
- **XML:** *Extensible Markup Language* – Lenguaje de marcas utilizado para almacenar datos en forma legible.
- **Time-Out:** Tiempo de espera establecido para la realización de una tarea.
- **SIFT:** *Scale-invariant feature transform* – Algoritmo de detección de objetos en imágenes.
- **Open source:** Programa cuyo código está disponible para el público para su modificación.
- **Símbolo del Sistema:** *(Command Prompt)* – intérprete de comandos para el sistema operativo Windows.
- **Lcc-win:** Compilador del lenguaje C para el sistema operativo Windows.
- **Trade-off:** Compensación existente entre dos características deseadas del programa.

1. INTRODUCCIÓN

1.1 MOTIVACIÓN

Dicen que una imagen vale más que mil palabras, por lo que encontrar una que represente lo que queremos decir es una tarea importante. Las imágenes facilitan la comprensión de los textos (atrayendo así más visualizaciones) y son capaces de captar la atención del lector ya que los hacen más interesantes.

Por ejemplo, es muy común encontrarse con documentos y artículos acerca del calentamiento global y sus consecuencias para la humanidad, pero sin una imagen en la que se vea el deshielo del ártico, los animales en extinción y otros efectos en la naturaleza, muchos lectores no llegarían a comprender la extensión y gravedad del problema. De una cierta manera, las imágenes muestran una realidad más allá de las palabras. Por esta razón, es deseable encontrar la imagen que haga que el lector siga leyendo y se interese más por el tema. Este es el caso para redactores de periódicos, blogs, o cualquier tipo de documento textual.

Al analizar un texto largo, nos damos cuenta de que muchas veces pueden llegar a tratarse distintos temas. Sin embargo, la tarea de encontrar imágenes que sirvan para representar un conjunto de temas es más compleja que cuando se trata de uno solo. Además el número creciente de imágenes en la red hace esta tarea aún más difícil.

Según un estudio[1] realizado en 1999, el número de imágenes indexadas en la web era de 180 millones, por lo que había aproximadamente 3Tb solo de datos de imagen. En 2012[2], 300 millones de imágenes se añadían a la red social FACEBOOK en un día, generando 7 petabytes de datos de imágenes cada mes. Sumando este número a los millones de páginas web, usuarios y organizaciones (ejemplo: NASA, genera miles de imágenes) que generan imágenes constantemente, vemos que hay un número impresionante de imágenes en la red y recorrer todas sería una tarea intolerablemente costosa.

De esta forma, buscar imágenes que puedan representar a más de un tema no es fácil, y es un proceso que implica la utilización de conocimientos/herramientas en las áreas:

- Theme-segmentation

[1] Accessibility of information on the web – Lawrence & Giles, 1999

[2] Internet 2012 in numbers - http://royal.pingdom.com/2013/01/16/internet-2012-in-numbers/

- CBIR (Content based image retrieval)
- IR (Information Retrieval)
- Image analysis
- Image similarity

Hay varios métodos para buscar imágenes asociadas a textos, por ejemplo, la extracción de palabras clave (*key words*). Pero si el texto es multi-tema, extraer palabras clave de todo puede generar cosas genéricas y poco descriptivas.

En este trabajo enfocamos el problema en una luz nueva, donde en lugar de derivar solo un conjunto de palabras clave, tratando todos los temas en un único descriptor, se separan los temas y se derivan palabras clave separadas por cada tema. La separación de los temas permite obtener conjuntos de palabras-clave separados y una búsqueda independiente de imágenes. Sólo al final las juntaremos todas, basándonos en su similitud visual, encontrando así un único conjunto que representa todo el documento.

1.2 OBJETIVO Y ESTRUCTURA

Con el fin de facilitar la búsqueda comentada en el apartado anterior, se plantea el desarrollo de un programa que para un determinado texto proporcione un conjunto de imágenes que lo describa, facilitando así la tarea del redactor. Al escribir un texto, una persona seguramente accederá a un motor de búsqueda para buscar imágenes. Si no se trata de un texto largo y el tema tratado está muy claro, su trabajo no será muy difícil. Sin embargo, al hacerse más largo incluye la tarea de recorrer miles de imágenes para cada posible tema que llegara a tratar su texto.

Frente a esto, el programa planteado encontrará un conjunto de imágenes en el que se pueda realizar la búsqueda. De esta forma en lugar de buscar entre miles de imágenes disponibles en la red, se puede buscar entre 20 o 30 seleccionadas.

Dado un texto largo en el que se tratan distintos temas, el primer paso que debemos realizar es la segmentación de dicho texto por tópicos, de manera que se obtengan distintos fragmentos (cada uno correspondiente a un tópico).

Cada uno de los fragmentos resultantes dará lugar a una lista de palabras-clave cada una con un peso asociado. Se asociará un peso mayor a las que se considerarán más específicas para el tema y por lo tanto mejores para la búsqueda. Con la lista de palabras-clave será posible obtener N

imágenes que representan al fragmento correspondiente, usando técnicas de IR para comparar los fragmentos con una descripción textual de las imágenes.

Por último, a través de medidas de similitud (*image similarity*) se busca una "intersección" de los N subconjuntos obtenidos, de manera que se obtengan las mejores imágenes que representen todos los temas.

Siguiendo este razonamiento, este trabajo se ha desarrollado dividiéndolo en 4 partes (módulos):

1. Segmentación del texto por tópicos (*Theme-based text segmentation*)
2. Análisis de cada fragmento de texto: Identificación de las palabras clave y cálculo de los pesos asociados.
3. Búsqueda de imágenes para cada fragmento textual
4. *Feature Extraction* – Comparación de imágenes
5. Determinación del conjunto "intersección" óptimo.

En el siguiente diagrama (*Figura 1*) se demuestra de manera resumida el proceso seguido por el texto desde que entra en el programa principal y pasa por los distintos módulos hasta que se seleccionan las mejores imágenes obtenidas.

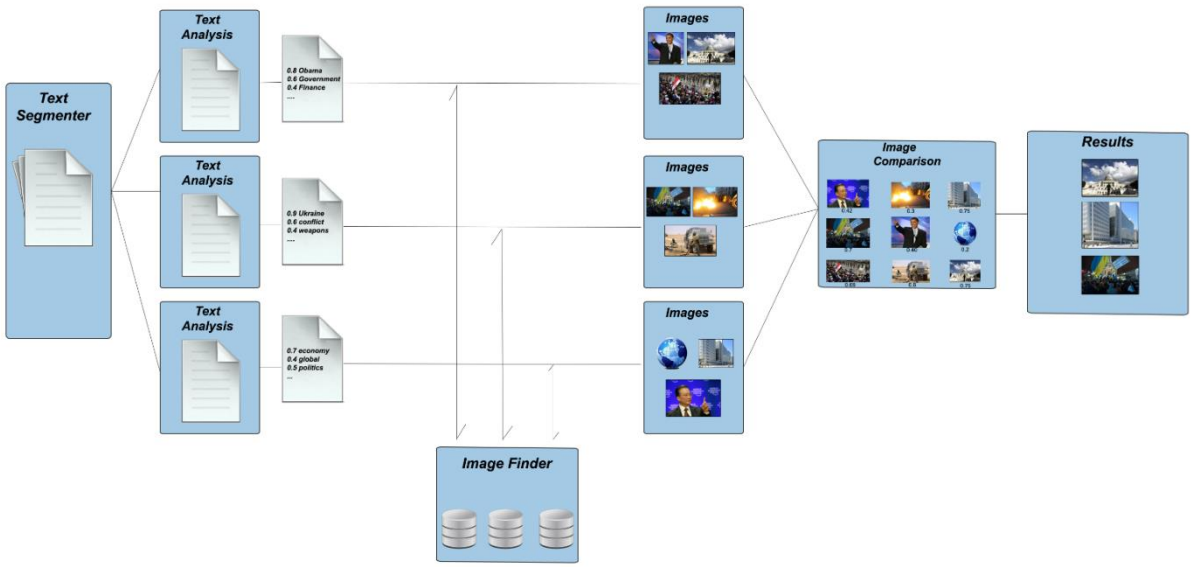


Figura 1 - Diagrama de conexión entre los distintos módulos

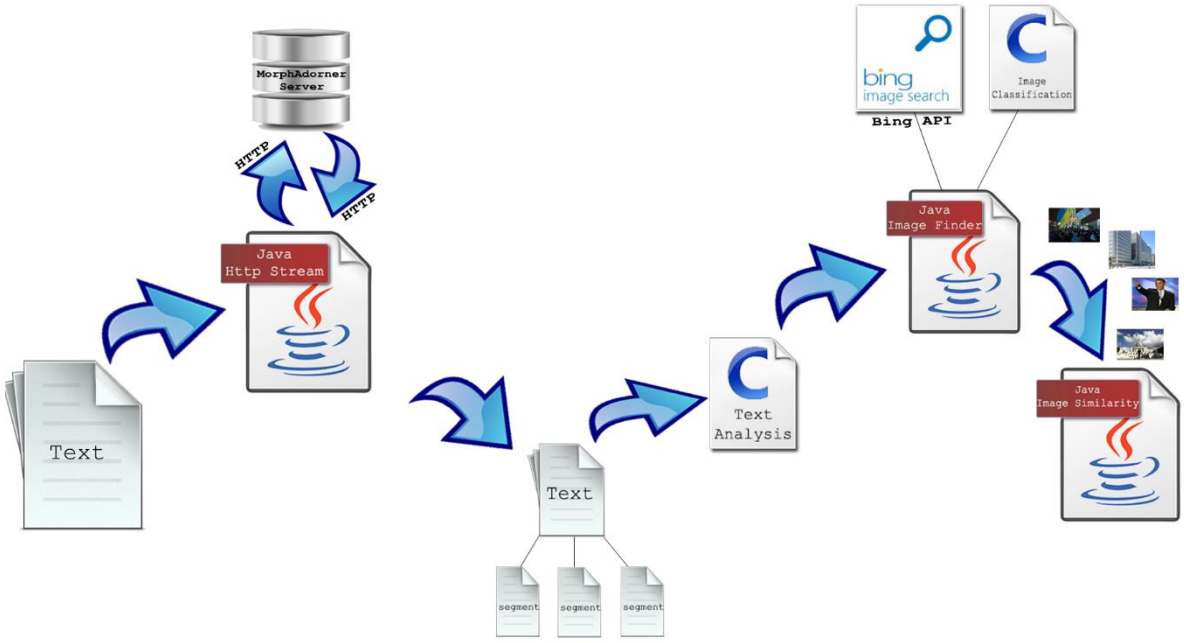


Figura 2 – Diagrama de la interacción de los programas desarrollado

1.3 ENTORNO DE DESARROLLO

Este proyecto se ha desarrollado en su mayor parte utilizando la herramienta Eclipse.

Eclipse [3] es un entorno de desarrollo multiplataforma que permite la implementación de aplicaciones a través de diversas herramientas. Se basa en la integración de *plug-ins*, proporcionando la API necesaria para que desarrolladores puedan implementarlos con enfoque en los servicios en los que desean especializarse. El programa aporta toda la estructura necesaria para incluir dichos *plug-ins* y determinar el código que debe ser utilizado en cada momento [21].

La implementación se lleva a cabo en el sistema operativo Windows, por lo que es necesario utilizar una herramienta que permita compilar los programas escritos en el lenguaje de programación ***C*** para que funcionen en dicho sistema operativo.

Lcc-Win [4] es un compilador de C para Windows que permite realizar esta tarea. Presenta dos versiones **lcc-win32** y **lcc-win64**, dependiendo del sistema operativo utilizado. Al ejecutarse, genera un archivo ejecutable (**.exe**) para el programa (**.c**) indicado.

[3] Página web de Eclipse - http://www.eclipse.org/

[4] Página web de Lcc-Win: http://www.cs.virginia.edu/~lcc-win32/

La última herramienta necesaria es la librería de procesamiento de imágenes **OpenCV**[5], que incluye cientos de algoritmos que facilitan las operaciones realizadas con imágenes para la realización de operaciones complejas. Originalmente desarrollada para C++, se ha extendido a otros lenguajes como C, Python y JAVA.

La información acerca de la instalación y configuración de las herramientas mencionadas se describe en el Anexo B.

1.4 PRUEBAS REALIZADAS

Para que se pueda apreciar mejor los resultados obtenidos, los ejemplos descritos a lo largo de este trabajo están basados siempre en el mismo texto de prueba. Dicho texto se ha obtenido a través de las páginas web de periódicos americanos, y está formado por fragmentos de tres artículos cuyo tema se especifica a continuación.

- Conflicto en Ucrania.
- Desaparición del avión de Malaysia Airlines.
- Conflicto en Siria y la utilización de armas químicas.

Se han elegido estos temas debido a su actualidad y porque presentan ciertas características en común. El texto completo, así como las referencias a los artículos desde el que se han obtenido se incluyen en el **Anexo A**.

1.5 ESTRUCTURA DEL DOCUMENTO

Este documento se estructura de la siguiente manera:

En el capítulo 2 se trata el tema de la segmentación de texto, proporcionando una breve introducción a la tarea de segmentación, una explicación de las herramientas utilizadas y el programa desarrollado, y los resultados obtenidos para el texto de prueba.

A continuación, en el capítulo 3 se detalla el proceso seguido para obtener las palabras-clave (keywords) de cada segmento y su peso asociado. Además, se describe el programa implementado para realizar esta tarea y algunos resultados obtenidos.

El capítulo 4 trata la búsqueda de imágenes para cada fragmento, a través de bases de datos ya desarrolladas o la API del motor de búsqueda Bing. Incluye una descripción de los métodos

[5] Página Web de OpenCV: http://opencv.org/

necesarios para realizar dicha búsqueda, establecer una conexión con los servidores correspondientes y la descarga de imágenes.

Luego, en el capítulo 5 se explica el método de comparación de imágenes desarrollado para este trabajo, que incluye la utilización de técnicas de procesamiento como comparación de histogramas y detección de bordes. También se incluyen las pruebas realizadas para un conjunto de imágenes y los resultados obtenidos según el método utilizado.

El programa principal encargado de realizar la conexión entre todos los módulos que componen el proyecto se describe en el capítulo 6, y el capítulo 7 incluye las pruebas finales realizadas y un análisis de los resultados obtenidos.

El documento finaliza con las conclusiones del proyecto, donde se resumen los puntos relevantes logrados en los apartados anteriores y el posible trabajo futuro que se podría realizar.

En el anexo se pone a disposición el texto de ejemplo utilizado en las pruebas del proyecto y un manual con la instalación y configuración de las herramientas utilizadas.

2. SEGMENTACIÓN DEL TEXTO

2.1 INTRODUCCIÓN A LA SEGMENTACIÓN DE TEXTO

La segmentación de texto puede ser definida como la identificación automática de los límites entre distintas unidades textuales (segmentos) en un texto [1]. Dichas unidades textuales se consideran segmentos del texto que presentan una coherencia temática (mismo tópico). Esta tarea puede ser útil en distintas áreas en las que se desea acceder a información más pequeña y coherente [2].

Actualmente existen muchos algoritmos y herramientas que consiguen segmentar un texto de manera eficiente y con un margen de error pequeño. El método utilizado en cada implementación varía, pudiendo hacer uso de técnicas como, por ejemplo, un análisis probabilístico del texto [2] o cohesión léxica [1].

En este trabajo se aplicará la segmentación de texto para conseguir una serie de fragmentos textuales que se refieran a distintos temas, con el objetivo de facilitar el análisis de las *keywords* en las siguientes partes, permitiendo así un análisis más efectivo que si analizáramos el texto completo. Esta tarea compone una parte importante del resultado ya que obtener los segmentos correctos es crucial para que las tareas realizadas posteriormente generen resultados correctos y coherentes.

El algoritmo de segmentación que utilizaremos es el C99 de Choi [4]. Se trata de un algoritmo de tres etapas que usa técnicas de procesamiento de imágenes para interpretar una representación gráfica de la similitud de cada par de sentencias del texto [1]. En este trabajo no se detallará el funcionamiento del algoritmo, ya que no es el enfoque principal. Sin embargo, si fuera necesario se puede consultar el documento de referencia [4].

En el siguiente apartado, se detallan las herramientas utilizadas en la implementación de esta primera parte. Los fragmentos obtenidos como resultado de este proceso se usarán en la segunda parte para su análisis individual.

2.2 DESARROLLO

Como se ha comentado anteriormente, el algoritmo de segmentación que se utilizará es el C99 de *Choi*. Se pueden encontrar diversas implementaciones de este algoritmo en la red. En este trabajo hemos realizado algunas pruebas preliminares con las herramientas *MorphAdorner*[6] y *UIMA Text Segmenter*[7], dos de las más conocidas actualmente, con el fin de determinar la que mejor se ajusta a nuestras necesidades.

Tras la instalación y el análisis de estas dos herramientas, se ha decidido utilizar *MorphAdorner*.

Esta elección viene dada por la precisión de los resultados con los textos probados y la "facilidad" de uso en comparación con las demás herramientas analizadas. No se han encontrado librerías (jar o ddl) para JAVA que permitan simplemente enlazar con el programa principal, por lo que se ha decidido utilizar un servidor Web.

2.2.1 - MORPHADORNER

MorphAdorner[3] es un programa JAVA de línea de comandos desarrollado por la *Northwestern Univeristy*, que permite realizar ciertas operaciones con textos tales como reconocimiento de sentencias, extracción de información, segmentación de texto, entre otros. Estas operaciones se pueden efectuar a través de la instalación de un programa cliente o un servidor, presentando incluso ciertas funcionalidades online.

En este trabajo se ha utilizado la versión SERVIDOR de *MorphAdorner* (*MorphAdorner SERVER* – un "servidor con protocolo HTTP" que incluye las funcionalidades de MorphAdorner en la web), ya que es la única que incluye la opción de realizar la segmentación de texto por HTTP con el método GET. La utilización de este método es necesaria pues facilita la conexión con el programa, ya que con *GET* todos los parámetros (incluyendo el texto que se va a segmentar) se pasan al servidor directamente en una URL generada en el programa principal.

La aplicación se descarga desde la página de la *Northwestern Univeristy* que incluye una guía de instalación.

Una vez instalada y ejecutada la aplicación del servidor, todos los servicios proporcionados estarán disponibles accediendo a ***http://localhost:8182*.**

[6] MorphAdorner - http://morphadorner.northwestern.edu/

[7] UIMA Text Segmenter - https://code.google.com/p/uima-text-segmenter/

La herramienta de segmentación con el método GET se encuentra en la dirección: http://localhost:8182/textsegmenter.html.

En esta página, tenemos la opción de configurar el proceso de segmentación según nuestra preferencia. La configuración utilizada se detalla a continuación. Esta ha sido elegida tras realizar una serie de pruebas con algunos textos y con el objetivo de optimizar la segmentación:

- ***Lexicon*:** Early Modern English.
- ***Segmenter*:** Aquí se ha seleccionado el algoritmo C99 con ***mask size*** de 11 e indicando que no sabemos el número de segmentos que hay en el texto (***Segments desired*** fijado a -1).
- ***Results Format*:** El resultado se va a obtener en formato XML, para facilitar su posterior análisis.

Para conectar este programa a la aplicación principal, es necesario el desarrollo de una función que obtenga la información y datos necesarios (texto a segmentar, parámetros del servidor), y los transforme en una oportuna llamada GET al servidor.

Con este fin, se ha desarrollado un pequeño programa en JAVA capaz de realizar esta tarea.

2.2.2 - APLICACIÓN JAVA PARA ACCEDER AL PROGRAMA DE SEGMENTACIÓN

Se ha elegido el lenguaje de programación JAVA para el desarrollo de este programa debido a la facilidad de manejo con técnicas de conexión HTTP. JAVA incluye clases ya programadas que facilitan la conexión con servidores *(java.net.HttpURLConnection)*, incluyendo métodos de creación de URL e intercambio de información.

El programa toma como parámetro el nombre del fichero donde se encuentra el texto que se desea segmentar.

Los métodos responsables de la conexión se encuentran en la clase *TextSegmenter.java*, que se ha desarrollado para este trabajo. Esta incluye los métodos necesarios para la conexión con el servidor, lectura del fichero de texto, formación de URL y almacenamiento de los resultados.

Antes de conectarse al servidor es necesario ejecutarlo. Gracias a las librerías de JAVA, eso se puede hacer directamente desde una función, utilizando un objeto de tipo *ProcessBuilder*, que permite ejecutar comandos del sistema operativo desde el propio programa JAVA.

La función *RunMAServer()* desarrollada se encarga de esta tarea, accediendo al directorio *"maserver-1.0.0"* (que se encuentra en la misma ruta que el programa principal), y ejecuta la aplicación del servidor (*runmaserver.bat*).

Como el servidor debe estar abierto durante todo el tiempo que se necesite sus servicios, dicha función se encarga de lanzar dos nuevos hilos para leer los buffers de entrada y error (*stdin* y *stderr*), permitiendo que la aplicación no se bloquee y que el servidor siga funcionando correctamente.

Una vez que el servidor se ha inicializado, el primer paso realizado por el programa es la formación de la URL. Como se está utilizando el método GET para la conexión HTTP, es necesario componer la URL de manera que pase todos los parámetros correctamente como parte de la misma URL siguiendo las convenciones de HTTP.

La estructura de la URL es la siguiente:

http:///localhost:8182/textsegmenter?text=

+ texto a segmentar
+*includeInputText=true&corpusConfig=eme&segmenterName=C99&c99MaskSize=11&c99SegmentsWanted=-1&tilerSlidingWindowSize=10&tilerStepSize=100&media=xml&segment=Segment*

La última parte hace referencia a los parámetros necesarios para la segmentación requeridos por *MorphAdorner*, como se comentó anteriormente. El texto a segmentar se lee directamente del fichero proporcionado y se sustituyen los espacios por el símbolo '+' en el proceso de formación de la URL. También, se eliminan ciertos caracteres reservados en http que puedan interferir en la URL al hacer la conexión, por ejemplo: '%'.

A través de un objeto de tipo *HttpURLConnection*, se envía la petición al servidor con la URL formada anteriormente, y se obtiene la respuesta.

La respuesta del servidor consiste en un texto XML que incluye información derivada del proceso de segmentación, como *tokens*, número de segmentos encontrados, y las partes del texto que componen cada segmento.

Obtener la respuesta en XML facilita la división del texto y la identificación de los segmentos. Cada segmento viene especificado entre las etiquetas <segmentText> y </segmentText>, por lo que una vez obtenida la respuesta solo es necesario separar el texto por estas etiquetas y recoger cada segmento.

Antes de finalizar el programa, se escribe cada uno de los segmentos en un fichero de texto distinto con el nombre: **"segmentID"**, en el que ID hace referencia al nº de segmento. Los ficheros se almacenan en el directorio *"TextAnalysis"*, presente en la misma ruta del programa principal, junto a los ficheros y ejecutables del programa que se encarga de analizarlos.

2.3 RESULTADOS

A continuación se dispone un fragmento de la respuesta XML devuelta por el servidor para el texto de ejemplo.

La única información que utilizamos de este fichero son los textos que componen cada segmento y que vienen indicados entre las etiquetas <segmentText> y </segmentText>. Analizando los segmentos encontrados, se puede observar que la herramienta consigue una segmentación aceptable. En nuestro ejemplo, el texto (que tiene, lo recordamos, tres temas distintos), es dividido en cuatro partes: la parte del texto que se refiere a Ucrania se ha dividido en dos segmentos pero los demás se identificaron en su totalidad.

```
<TextSegmenterResult>
      <text>
            "What's next in eastern Ukraine? Votes held in eastern Ukraine…"
      </text>
      <corpusConfig>eme</corpusConfig>
      <c99MaskSize>11</c99MaskSize>
      <c99SegmentsWanted>-1</c99SegmentsWanted>
      <tilerSlidingWindowSize>10</tilerSlidingWindowSize>
      <tilerStepSize>100</tilerStepSize>
      <sentences>
            <sentence>
                  <token>"</token>
                  <token>What's</token>
                  <token>next</token>
                  <token>in</token>
                  <token>eastern</token>
                  <token>Ukraine</token>
                  <token>?</token>
            </sentence>
            <sentence> …
            </sentence>
            <segments>
```

```
                <int>0</int>
                <int>19</int>
                <int>21</int>
                <int>31</int>
            </segments>
            <segmenterName>C99</segmenterName>
            <segmentTexts>
                <segmentText>
                "What's next in eastern Ukraine? Votes held in eastern
Ukraine …"
                </segmentText>
                <segmentText>
                "Such an autonomous state would have "a veto on Ukrainian
moves …"
                    <segmentText>
                    "Malaysia military says plane turned back Malaysian
officials backed away …"
                    </segmentText>
                    <segmentText>
                    "U.N. report concludes that chemical weapons were
used in Syrian conflict …"
                    </segmentText>
            </segmentTexts>
</TextSegmenterResult>
```

CAPÍTULO 3 - ANÁLISIS DE CADA FRAGMENTO DE TEXTO

3.1 INTRODUCCIÓN A *INFORMATION RETRIEVAL*

El tema que tratamos en esta parte forma parte del campo general de la IR (*information retrieval*). *Information Retrieval* consiste en la búsqueda y recuperación de determinada información en un conjunto de datos no ordenado. En los últimos años su importancia ha ido creciendo ya que es usada todos los días en motores de búsqueda y correos [5].

Existen varios modelos de IR actualmente, siendo los más importantes el Modelo Vectorial y los modelos probabilísticos [6].

En el modelo vectorial, el texto se representa como un vector de términos (normalmente palabras), donde cada uno corresponde a una dimensión en un espacio vectorial de muchas dimensiones. Cada término se asocia con un peso, permitiendo ordenar los resultados como un ranking. Los modelos probabilísticos usan técnicas de probabilidad para calcular la probabilidad de relevancia de un término en el documento. [6]

En este trabajo se utiliza el modelo vectorial. A partir de este momento, al referirse a "texto", se está refiriendo a un fragmento textual obtenido en el apartado anterior. Todo el proceso descrito en esta fase y en las siguientes se repite para cada segmento encontrado.

Los fragmentos obtenidos se tratan como una unidad independiente. Se analiza cada uno con el objetivo de obtener una lista de palabras con un peso asignado a cada una. No se tienen en cuenta en esta lista las llamadas ***stop words***, es decir, palabras habituales en el vocabulario que no aportan información al texto [5], por ejemplo: "*the, to, a, an*, etc.". Dichas palabras son filtradas antes del análisis con el fin de mejorar este proceso.

Muchas de las características y métodos de análisis aquí comentados dependen del idioma que se está utilizando. En este caso, todo se ha basado en la lengua inglesa debido a la cantidad de información relacionada disponible para este idioma y la facilidad de acceso a librerías ya desarrolladas.

Por motivos gramaticales, los documentos de texto normalmente emplean diferentes formas de una palabra. Además, existen familias de palabras relacionadas que tienen un significado semejante, por lo que se busca una manera de reducir estas variaciones a una base común [5].

Por ejemplo, para el siguiente grupo de palabras (*man → men, book → books, study → studied/studying*), todas las formas derivadas mantienen el significado de la palabra original, pero debido a estas pequeñas variaciones en tiempos verbales, plurales, etc., puede dificultar su comparación.

Para mejorar el análisis del texto, se aplica una serie de técnicas que facilitan la identificación y comparación de palabras:

Stemming

Proceso que elimina el final de las palabras con el objetivo de transformarla a una base común [5]. La palabra resultante es conocida como ***Stem***. Ejemplos:

Palabra	Stem
Studying, studied	*studi*
Government	*govern*
Books	*book*

Tabla 1 – Ejemplos de Stemming

A partir de este momento al referirse a "palabras", se está refiriendo a ***Stem***, una vez que para todas las palabras leídas se usará su ***stem***.

A continuación se transforman todas las letras de la palabra a minúsculas para facilitar la comparación [5], en un proceso conocido como ***Capitalization/case-folding***.

La última etapa de este proceso consiste en la eliminación de caracteres y acentos de las palabras, ya que pueden perjudicar la comparación. Ejemplo: "*he's → he. Government's → government*"

Tras realizar pruebas, también se ha incluido la eliminación de números, ya que estos no añaden información relevante a la búsqueda de imágenes en el siguiente paso.

Por ejemplo: Aunque se puede dar el caso de que el número se refiera a una fecha importante en la historia (ejemplo: 1945 podría estar relacionado a sucesos como la segunda guerra mundial), otros no traen ninguna información, además de introducir más incertidumbre al proceso (ejemplo: edades).

Una vez acabado este proceso de mejoras en las palabras, se procede al cálculo de los pesos. Para ello, se hace uso de un ***Corpus*** que contiene N palabras con un **peso** correspondiente,

relacionado a su frecuencia en el conjunto de la lengua inglesa. Palabras más específicas tendrán pesos más altos, mientras algunas más comunes tendrán pesos más bajos.

El peso de cada palabra indica la importancia de esa palabra como característica del texto. Dicho peso viene dado por la siguiente fórmula:

$$v_i = fd_i * \log(1/f_i)$$

Ecuación 1 – Cálculo del peso de una palabra

Donde:

fd_i es el número de veces que aparece la palabra en el texto analizado.

f_i es la frecuencia de la palabra analizada en el ***Corpus*** comentado anteriormente. En el caso de que la palabra no se encuentre en el corpus, se asigna una frecuencia muy baja, inferior a todas las palabras del *corpus*.

De esta forma se puede observar que un peso alto está asociado a palabras que aparecen muchas veces en el texto analizado y pocas en el *Corpus*. Por ejemplo:

- Si tenemos 10 ocurrencias de la palabra *"man"* en el texto, esta no será considerada muy importante, al ser una palabra muy común y en consecuencia tener un peso (**W_i**) más bajo.
- A su vez, si la palabra *"fluorescence"* aparece 10 veces, esta es considerada mucho más importante, al no ser una palabra común y en consecuencia tendrá un peso (**W_i**) más alto.

Un texto con stem s_1 ……. s_N da lugar a un vector de pesos (no normalizados) [V_1 …… V_N].

Por último se normalizan todos los pesos, de manera que la frecuencia de cada palabra se divide entre la raíz cuadrada de la suma de los cuadrados de todos los pesos.

$$w_i = v_i / \sqrt{\sum_k v_k^2}$$

Ecuación 2 – Normalización de los pesos de cada palabra

Se usan técnicas para evitar representar todo el vector, de manera que se forma una lista de pares: palabra,peso - (s_i, w_i), que es la que se va a escribir en un fichero para su posterior procesamiento en el siguiente paso.

3.2 DESARROLLO

Esta etapa se ha realizado a través de la implementación de un programa en C, que realiza las funciones correspondientes. Se ha elegido este lenguaje porque ya teníamos una librería implementada en C que gestiona la parte de *stemming* de las palabras y el manejo de las *stopwords*.

Para dicha implementación se han desarrollado/utilizado los siguientes ficheros:

- *TextSegmenter.c*: Programa principal que llama a las funciones necesarias para el funcionamiento correcto. Recibe como parámetros:
    - Nombre del fichero donde se encuentra el texto que queremos analizar.
    - Nombre del fichero que contiene las "*Stop words*".
- *TextAnalysis.c y TextAnalysis.h*: Librería desarrollada para gestionar la identificación de las palabras del texto y calcular las frecuencias.

  Además, se han definido estructuras que almacenan los datos utilizados.
    - ***Stem***: Almacena el *stem* leído (*string*) y la frecuencia (*double*).
    - **StemList**: Lista de *Stems*.
    - ***Wfreq***: Misma estructura que *Stem*, pero se usa para almacenar las palabras del corpus y sus frecuencias.
    - ***Wlist***: Lista de palabras del corpus, con el total de las palabras leídas y la frecuencia total.
- *Stem.h* y *Stem.c*: Librería auxiliar que disponíamos y que incluye funciones para obtener el *stem* de una palabra.
- *Stopword.c* y *Stopword.h*: Librería auxiliar que disponíamos y que incluye funciones para gestionar las *"stop words"* de un fichero.

Algoritmo

El programa funciona de la siguiente manera:

1. Carga la lista de s*top words* en la estructura

2. Carga el *corpus* en la estructura.
3. Para cada palabra leída del fichero
    a. Elimina caracteres no deseados.
    b. Comprueba si es una *stop word*.
    c. Si no es una *stop word*, obtiene el *stem* y lo almacena en la lista.
    d. Si el *stem* ya estaba en la lista, incrementa la frecuencia.
4. Una vez almacenadas las palabras, se realiza un cálculo de todas las frecuencias siguiendo la ecuación 2 en el apartado anterior.
5. Por último, escribe todas las palabras y su peso en un fichero para que pueda ser utilizado en el siguiente paso.

3.3 - RESULTADOS

En la tabla 2 se incluyen las 10 primeras palabras más importantes y sus pesos, obtenidos en el fichero de análisis de cada segmento del ejemplo que se está utilizando.

Segmento 1 (Segment1Analysed.txt)	**Segmento 2** (Segment2Analysed.txt)	**Segmento 3** (Segment3Analysed.txt)	**Segmento 4** (Segment4Analysed.txt)
Ukrain – 0.462788	**European - 0.278674**	**radar - 0.370876**	**Syrian - 0.368317**
Separatist – 0.27411	**Putin - 0.265774**	**plane - 0.249387**	**weapon - 0.347214**
Govern – 0.236113	**facto - 0.265774**	**turn - 0.23329**	**report - 0.306691**
Bugajski – 0.23346	**Obama - 0.265774**	**militari - 0.224851**	**chemic - 0.293446**
Elect – 0.213316	**Damon - 0.224523**	**Malaysia - 0.221466**	**Ghouta - 0.246736**
Kiev – 0.187423	**Ukrainian - 0.19974**	**Malacca - 0.200332**	**evid - 0.16852**
Russia – 0.181911	**veto - 0.195703**	**Rahman - 0.200332**	**Sellstrom - 0.164491**
Eastern – 0.176005	**autonom - 0.185902**	**flight - 0.180065**	**Mission - 0.158879**
Central – 0.153402	**interfer - 0.174261**	**Malaysian - 0.16154**	**conclud - 0.149232**
East – 0.125178	**Moscow - 0.171277**	**todai - 0.156843**	**attack - 0.13189**

Tabla 2 – Resultados: Análisis de segmentos

Podemos observar que los resultados coinciden con lo que se podría esperar tras leer el texto utilizado. Las palabras de mayor peso, y que aparecen en los 5 primeros puestos de cada

segmento, son muy específicas del tema tratado, por lo que no debería suponer un problema a la hora de buscar las imágenes.

CAPÍTULO 4 - BÚSQUEDA DE IMÁGENES PARA CADA FRAGMENTO TEXTUAL

Una vez obtenida la lista de palabras y pesos de cada fragmento, se buscan imágenes que lo representen. Para ello, es necesario realizar una búsqueda en una base de datos con muchas imágenes, de manera a encontrar las que mejor lo representan.

La idea inicial para esta parte era utilizar una base de datos de imágenes (1 millón de imágenes) que ya teníamos, pero tras el análisis de los resultados, se ha incluido una segunda opción que se detalla más adelante.

4.1 BÚSQUEDA EN BASE DE DATOS CON ETIQUETAS

La primera opción implementada consiste en utilizar una base de datos de 1 millón de imágenes que ya disponíamos, que incluye la URL de cada imagen y un conjunto de palabras (***captions***) que la describen brevemente.

La búsqueda se realiza a través de un producto escalar de dos vectores: El vector compuesto por los pares (palabra, peso) del texto analizado previamente, y el vector formado con la información de la base de datos.

Procesamiento de imágenes de la base de datos

Antes de realizar la búsqueda de las imágenes, procesamos la base de datos para conseguir una lista que facilite esta tarea. En este caso, y como la base de datos no es excesivamente grande (1 millón de imágenes), cada entrada de la lista contiene la URL de la imagen y un conjunto de pares (palabra, peso) obtenidos a partir de las *captions* de cada una.

Búsqueda de imágenes

En esta parte del trabajo ya se ha obtenido la lista de pares (palabra, peso) del texto, y una segunda lista con la información de las imágenes de la base de datos.

De esta manera, para encontrar las imágenes que mejor representen el texto, se busca una **medida** de similitud, donde valores más altos corresponderán a las mejores imágenes y los más bajos a las peores.

El valor seleccionado para este trabajo es el **producto escalar** de los dos vectores. Es decir, se calcula el producto escalar del vector del texto con el vector de las imágenes. Esta operación solo funciona pues los vectores han sido normalizados previamente.

Este producto consiste en la suma del producto de los pesos de las palabras que se encuentren tanto en el vector del texto, como en la lista de *captions* de la imagen. El valor obtenido nos ofrece una buena indicación de las mejores imágenes.

Una vez obtenidas las N mejores imágenes para cada texto, se pasa a la última parte, donde habrá que buscar las mejores de la intersección de los N fragmentos.

4.2 BÚSQUEDA AUXILIADA POR BING SEARCH API

El problema encontrado con la búsqueda en la base de datos de la opción 1 fue la calidad de las imágenes recuperadas para los temas estudiados: dado que la base de datos es relativamente vieja y no se ha actualizado, no contiene imágenes o etiquetas relevantes para ciertos temas de actualidad. Por ejemplo: Si hacemos una búsqueda acerca de un evento que ha sucedido en los últimos meses, como la desaparición del avión de Malaysia Airlines, la base de datos nos devolvería con suerte imágenes de cualquier avión, mar, etc.

Frente a este problema y con el fin de mejorar la calidad del resultado del programa final, se han buscado alternativas. La mejor opción encontrada fue utilizar una API de un motor de búsqueda para enlazar con el programa. Las dos APIs estudiadas más destacadas son la de GOOGLE y la de BING.

Finalmente, se ha decidido utilizar la API de Bing ya que ofrece planes de suscripción gratis y los métodos de conexión y obtención de información son bastante sencillos.

Bing search API

Bing[8] es un motor de búsqueda desarrollado por Microsoft, que permite la búsqueda de información, imágenes, videos, noticias y otros servicios relacionados. Microsoft proporciona diversas APIs para que desarrolladores puedan utilizar los servicios que proporcionan en sus aplicaciones. En este trabajo, se utiliza la API de búsqueda (Bing Search API).

Bing Search API es una API que permite a los desarrolladores integrar los servicios de búsqueda de contenido, imágenes, noticias y videos a su aplicación usando XML o JSON [7]. La plataforma presenta distintos tipos de suscripción, para que se elija la que mejor se adapta a nuestras necesidades. Cada una de ellas viene definida por el número de transacciones mensuales que se

[8] Bing: https://www.bing.com/

permite realizar con la base de datos de BING. Para el objetivo de este trabajo, se ha suscrito a la opción gratis que da derecho a 5.000 transacciones al mes.

Datos de interés

Para acceder al servicio es necesario tener una cuenta de Microsoft. Una vez dado de alta e introducidos los datos de Idioma, el usuario obtiene una clave de identificación única, que sirve para validar su acceso cada vez que se hagan transacciones entre la aplicación y Bing. Además, el usuario tiene acceso a un menú donde se incluyen datos de uso.

Figura 3 – Bing Search API Control center

Consultas

Todas las consultas se realizan a partir de la URI: https://api.datamarket.azure.com/Bing/Search, a la cual se añaden los parámetros y datos deseados. Bing proporciona una serie de parámetros para personalizar la búsqueda: **[8]**

- El primer parámetro que se añade es la categoría de la consulta (Imagen, video,…). En este caso sólo queremos encontrar Imágenes, por lo que se añade */Image* a la URI base.
- A continuación se incluye la consulta que se va a realizar, entre comillas simples siguiendo el formato "*?Query='consulta'*.
- También se puede incluir un parámetro (**$top**) que indica el número máximo de resultados que se desea obtener. Para nosotros este valor será N + 3, donde N es el número máximo de imágenes que queremos comparar por segmento. Se recogen 3 imágenes adicionales, ya que algunas veces una o más URLs obtenidas pueden hacer referencia a una imagen que ya no está disponible. De esta forma, si se diera el caso, no quedarían menos imágenes.

Además, existen otros parámetros que permiten definir el tamaño de las imágenes deseadas, país de procedencia de los resultados, etc. En este caso, no se van a usar estos parámetros, ya que las imágenes serán redimensionadas en la parte de comparación, y, de esta manera, con menos filtros se obtienen más resultados relevantes.

El identificador obtenido al registrarse es necesario para poder establecer una conexión con la URI y realizar la consulta. Cada consulta enviada cuenta como una transacción, disminuyendo la cantidad de transacciones disponibles contratadas.

La respuesta se puede obtener en formato XML o JSON. Aquí, utilizamos únicamente XML debido a la facilidad de recuperación de las URLs. El fichero devuelto contiene los siguientes datos:

Nombre	Tipo	Descripción
ID	Guid	Identificador
Title	String	Título de la imagen
MediaUrl	String	URL de la imagen en tamaño original
SourceUrl	String	URL de la página web que contiene la imagen
DisplayUrl	String	URL que aparece en la página de resultados de la búsqueda
Width	Int32	Anchura de la imagen, en pixeles, si está disponible
Height	Int32	Altura de la imagen, en pixeles, si está disponible
FileSize	Int64	Tamaño de la imagen, en bytes, si está disponible
ContentType	String	MIME de la imagen, si está disponible
Thumbnail	Thumbnail	Propiedades de la miniatura de la imagen

Tabla 3 – Datos devueltos al realizar la consulta [8]

La consulta se realiza inicialmente con las 5 palabras más importantes (mayor peso) del texto analizado. Se seleccionan únicamente 5, ya que al incluir mucha información, puede perjudicar los resultados de la búsqueda, por lo que, según las pruebas realizadas, esta cantidad debería proporcionar un buen resultado.

Se han observado algunos casos en que este número de palabras no ha logrado obtener ningún resultado o ha encontrado muy pocas imágenes. Esto se produce en casos donde las 5 palabras de mayor peso son muy específicas y, al juntarlas, la búsqueda se hace difícil. Para intentar solucionar este problema y evitar que algunos segmentos tengan muy pocas imágenes disponibles, se establece un umbral de "*MAX_RANKING" – 5*, donde MAX_RANKING es el número máximo de imágenes deseadas. Si tras realizar la búsqueda no se obtienen suficientes resultados, se vuelve a realizarla eliminando la última palabra (manteniéndose las 4 más importantes). Este proceso se repite (si necesario) hasta que queden las 2 palabras más importantes.

Esto difiere de la primera opción, en la que todas las palabras del texto analizado se comparaban con la descripción de la imagen.

Una vez obtenidas las URLS, se descargan todas las imágenes para cada segmento en un directorio distinto, que será usado posteriormente para la comparación.

4.3 DESARROLLO

Para obtener el resultado anterior, es necesario desarrollar una función que realice las operaciones necesarias. Con este fin, se han desarrollado dos programas. El primero de ellos (escrito en C) se usa solo en el caso de que se vaya a utilizar la **Opción 1** (*captioned-database*).

4.3.1 - PRE-PROCESAMIENTO DE LA BASE DE DATOS (LENGUAJE C)

El primer programa se ha desarrollado en C ya que se reutilizan algunas funciones de las librerías desarrolladas en este lenguaje en el módulo de análisis del texto (*stemming*, *stopwords*, cálculo de frecuencias). Dicho programa se encarga de procesar la base de datos y obtener la lista de *captions* y URLS con pesos.

La base de datos está compuesta de dos ficheros de texto (*ioned_photo_dataset_urls.txt*, que contiene una URL por línea, e *ioned_photo_dataset_captions.txt*, que incluye en cada línea las *captions* de cada imagen del otro fichero).

El objetivo del programa es leer cada una de las líneas de los ficheros, almacenar las *captions* en una estructura "StemList" (como en el paso de Análisis) y calcular la frecuencia para cada palabra de la *caption*. Por último, se escribe en un fichero de salida la información obtenida para cada imagen.

Cada línea del fichero de salida corresponde a una imagen y sigue el siguiente formato:

Npalabras (palabra peso) …… URL

Es decir, el primer número indica el número de palabras que vienen a continuación (obtenidas en las *captions*). Luego, vienen N pares (palabra, peso), y por último la URL.

El algoritmo seguido por el programa es el siguiente:

1. Carga el Corpus (El mismo que se ha comentado en el apartado anterior).
2. Por cada imagen:
    a. Lee la URL del fichero de URL's.
    b. Lee las *captions* del fichero de *captions*.
    c. Crea una *StemList* con las palabras leídas siguiendo el proceso de análisis (*stemming*, eliminación de caracteres especiales, eliminación de *stopwords*, etc.).
    d. Calcula la frecuencia de todas las palabras de la *StemList*.
    e. Escribe los datos obtenidos en el fichero de texto siguiendo el formato comentado.

El programa está compuesto por los ficheros:

- *imageAnalysis.c* e *ImageAnalysis.h*: Librería desarrollada con las funciones necesarias para realizar el procesamiento descrito. Se ha definido una estructura auxiliar llamada ***Image***, donde se almacena la URL (string) y la lista de *stems* (StemList, tipo definido en la parte de análisis).
- *imageClassification.c*: programa principal encargado de llamar a las funciones necesarias. Tiene como parámetros:
    - nombre del fichero de URL's
    - nombre del fichero de *captions*
    - nombre del fichero de StopWords que se va a usar

El fichero de salida se llama **imageDatabase.txt**. Este programa solo necesita ejecutarse en el caso de que se haya actualizado algunos de los ficheros de la base de datos, por lo que normalmente se puede hacer uso directamente del fichero **imageDatabase.txt** generado. Este

fichero es la base de datos a la que se van a realizar las consultas de imágenes, suponiendo que queramos utilizar el método 1 y no el de BING.

4.3.2 - BÚSQUEDA DE IMÁGENES (JAVA – IMAGE FINDER)

Para la búsqueda de imágenes se ha desarrollado una librería en JAVA, por su facilidad de manipulación de conexiones y descargas de imágenes, lo que supone una parte importante.

La librería está implementada en la clase ImageFinder (ImageFinder.java), donde se incluyen las funciones de descarga de imágenes, conexión a BING API, selección de imágenes, etc. Recibe como parámetros una variable (*boolean*) que indica si se va a utilizar la API de Bing o no (true si se va a usar bing), y el nombre del fichero en el que se encuentra la base de datos (si se va a utilizar la base de datos ya obtenida a partir del programa en C).

El método principal de esta librería (*ImageFinderMain*) recibe la ruta del directorio donde están almacenados los segmentos de texto analizados en el paso 2, y el número de segmentos encontrados. Dicho método sigue el siguiente algoritmo:

Para cada segmento de texto existente

- Crea un nuevo directorio siguiendo la nomenclatura "*segment**ID**images*", donde ID hace referencia al número del segmento. Si el directorio ya existiera, se borraría todo su contenido.
- Abre el fichero que contiene la lista de *(palabras,peso)* del texto analizado (*"segment**ID**Analysed.txt"*), y almacena localmente la *StemList*.
- Si la búsqueda se hace a través de la Bing API:
    - Obtiene las 5 palabras más importantes de la lista (mayor peso) que se van a utilizar en la consulta.
    - Conecta a BING siguiendo el protocolo que se explica más adelante.
    - Obtiene la respuesta de BING y almacena las N URLS, asignando un peso creciente a cada una, siguiendo el orden en que aparecen en el fichero XML devuelto.
- Si se hace en la base de datos procesada en C:
    - Lee y almacena localmente la lista de URLS y *StemList* de la base de datos.
    - Calcula el producto escalar de esta lista con la *StemList* del fichero de texto analizado.
    - Almacena las N URLS deseadas asignando un peso que será el producto escalar obtenido.
- Se descargan todas las imágenes en el directorio creado inicialmente.

Desarrollo de la búsqueda con *Bing Search API*

Para realizar la búsqueda usando la API de Bing, es necesario seguir todas las directrices establecidas por la compañía, ya que cualquier dato incorrecto puede afectar el funcionamiento.

El primer paso realizado es obtener las palabras que se van a utilizar en la consulta. Para ello, cada par de (palabra,peso) leídos del fichero de texto analizado se introducen en una cola de prioridad (clase *PriorityQueue<>* en JAVA).

```
PriorityQueue<Word> words = new PriorityQueue<Word>(maxRanking,
                                Collections.reverseOrder(wordComparator));
```

Esta se crea con un comparador personalizado (*WordComparator*) implementado para indicar a la cola cómo se comparan las palabras (*Word.weight*), y *maxRanking* indica el tamaño inicial de la cola.

Por defecto, las colas de prioridad en JAVA consideran que el menor peso tiene más prioridad, por lo que, como en este caso las de mayor peso tienen mayor prioridad, se usa el argumento *Collections.reverseOrder*, que se encarga de almacenarlas de mayor a menor.

Para obtener las mejores palabras solo es necesario extraer (*"pull"*) de la cola las N primeras palabras.

La conexión a bing se realiza a través de los métodos *connectToBingAPI y bingMatch*.

connectToBingAPI

El método se encarga de formar la URL y establecer los parámetros correctamente.

La consulta a realizar (palabras más importantes separadas por un espacio) se codifica al formato URL (*URLEncoder*), y se concatena a la URI base siguiendo la estructura que ya se ha comentado. El parámetro ***$top*** se incluye al final de la URL indicando que se desea obtener únicamente *maxRanking+3* resultados.

La clave de identificación del usuario debe ser codificada utilizando el sistema de codificación *Base64* (*Base64.encodeBase64* en java) siguiendo el formato: **AccountKey : AccountKey.**

Por último, se realiza la conexión con un objeto *URLConnection*, estableciendo la identificación con la propiedad *SetRequestProperty*, y obteniendo la respuesta a través de un Buffer e InputStream.

La respuesta es devuelta al método **bingMatch**, que obtiene las URLS. Esto se hace con un simple ***Split*** de la cadena obtenida, utilizando como delimitadores "<d:MediaUrl m:type="Edm.String"> y </d:MediaUrl>.

Como cada resultado de la búsqueda contiene dos etiquetas *MediaUrl* para cada imagen (una para el tamaño completo y otra para la miniatura), se eliminan las URLS obtenidas que contengan el texto: "pid=15.1", que hace referencia a las miniaturas. Todas las URLS obtenidas se almacenan en una cola de prioridad "ranking", cuyo peso será creciente según el orden en que vengan en el fichero devuelto.

Desarrollo de la búsqueda utilizando la base de datos: imageDatabase.txt

Esta modalidad de búsqueda se implementa en el método *LocalDatabaseMatch*, que lee todo el fichero de URLS de imágenes formando un vector y aplicando el producto escalar con el vector obtenido a partir de la lista de palabras del texto analizado (stemList).

El producto escalar se obtiene sumando el producto de los pesos de las palabras que aparecen tanto en la descripción de la imagen (*captions*) como en la lista obtenida del texto. Las imágenes de mayor peso se almacenan en la cola de prioridad ***ranking***. En la tabla 4 se muestra un ejemplo de este proceso para el segmento 1 obtenido a partir del texto de ejemplo y una de las URLS que se encuentran en la base de datos.

Imagen	**StemList obtenida a partir del texto**	**Producto (w_i)**
URL 1 – **Girl** 0.502177 **Field** 0.507906 **Ukrain** 0.699893	**Ukrain** 0.462788 **Govern** 0.236113 **Separatist** 0.27411 …	0.699893 * 0.462788 = 0.32390

Tabla 4 – Ejemplo de cálculo del peso para las imágenes de la base de datos

La búsqueda en este método puede llegar a tardar bastante, ya que hay que comparar todas las imágenes de la base de datos (1 millón para la base de datos actual) para cada uno de los segmentos. Con el fin de optimizar esa búsqueda y evitar tener que almacenar localmente 1 millón de (URL,peso) en la cola de prioridad, se sigue el siguiente algoritmo de comparación:

- Se introduce en la cola una URL vacía con peso 0, que será el mínimo actualmente en la cola. Como sabemos que queremos un máximo de *maxRanking* (20) imágenes, podemos insertar imágenes en la cola solo en el caso de que el peso supere el mínimo.
- Para cada una de las imágenes de la base de datos:
    - Calcula el producto escalar.
    - Si el producto es mayor que el peso mínimo actualmente en la cola:
        - Si la cola está llena (maxRanking): Elimina el actual mínimo, inserta la nueva imagen en la cola y calcula el nuevo mínimo (será el primero de la cola de prioridad)
        - Si no está llena, simplemente inserta la nueva imagen.

En la siguiente imagen, se muestra un ejemplo del algoritmo.

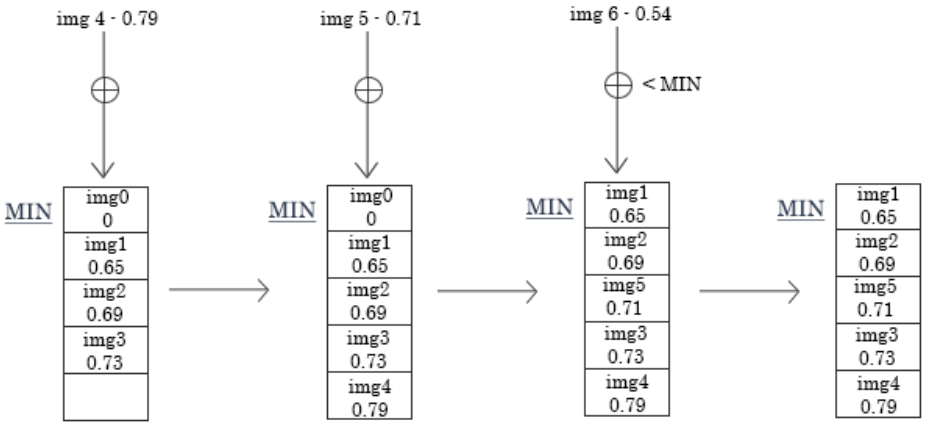


Figura 4 – Ejemplo del algoritmo de inserción utilizado en la búsqueda de imágenes

Descarga de imágenes

Por último, se descargan todas las imágenes seleccionadas en el directorio correspondiente al segmento. La descarga se realiza a través del método ***writeImageInFile***, que establece una conexión con la URL a través de los objetos *URLConnection* [9] y *HttpURLConnection* y descarga los datos.

Durante este proceso, se ha encontrado un problema cuando la URL obtenida hacía referencia a una imagen ya no válida (página no encontrada o redirección de URL). La solución encontrada fue que antes de descargar los datos, el programa recogiera el código de Respuesta HTTP (*httpConnection.*getResponseCode), y eliminara esa imagen de la lista en el caso de que el valor recibido fuera 404 (error) o 302 (redirección).

La solución al problema anterior añadía un nuevo problema, ya que la petición del código de respuesta es bloqueante, es decir, bloquea el programa hasta que se obtenga la respuesta o que

la operación devuelva *TIME-OUT*, lo que para JAVA es un tiempo de espera muy largo (alrededor de 3 minutos).

Para mejorar el tiempo de respuesta y evitar que afecte el tiempo de ejecución total del programa, se redefine el tiempo de *Time-Out* de lectura (*ReadTimeOut)* y conexión (*ConnectTimeOut)* para 30 segundos. Se han obtenido buenos resultados con el tiempo establecido.

Por esta razón, al obtener las URLS en los métodos anteriores, se obtienen 2 ó 3 adicionales, evitando que un segmento tenga menos imágenes debido a fallos de este tipo.

También se comprueba que la imagen se ha descargado correctamente, abriéndola mediante un objeto *Mat* de la librería de procesamiento de imágenes *OpenCV* (explicada con más detalles en el siguiente apartado), y asegurando que el número de pixeles (filas x columnas) es mayor que 0. Si no se da el caso, se elimina dicha imagen.

4.4 RESULTADOS

Los resultados obtenidos en esta parte han variado según el método utilizado. Inicialmente, solo se tenía pensado utilizar el método 1, pero como se comentó anteriormente, los resultados proporcionados no eran muy actuales y a veces podrían salir imágenes que no estaban directamente relacionadas con el tema.

Al introducir el método 2 con la API de BING, se ha visto una mejora considerable de las imágenes obtenidas. Esto se debe sobre todo al hecho de que la base de datos de BING es bastante más grande y actualizada que la nuestra, permitiendo obtener resultados mejores.

En las tablas 5 y 6 se pueden ver algunos de los resultados obtenidos para cada segmento del texto de ejemplo, según el método de búsqueda utilizado. Debido al número de imágenes descargadas (20), solo se incluirán 4.

Se puede observar que el primer método proporciona resultados bastante más actuales y esperados que el segundo, mejorando la fiabilidad del programa.

Segmentos	Resultados			
Segmento 1				
Segmento 2				
Segmento 3				
Segmento 4				

Tabla 5 – Resultados: Búsqueda de imagen con BING

Segmentos	Resultados			
Segmento 1				
Segmento 2				

Segmento 3				
Segmento 4				

Tabla 6 – Resultados: Búsqueda de imágenes con Base de Datos

Una vez acabada esta etapa, ya tenemos un conjunto de imágenes que representan cada uno de los segmentos de texto encontrados, por lo que se pasaría a la última parte para tratar de encontrar la "intersección" del conjunto.

5. IMAGE SIMILARITY

5.1 INTRODUCCIÓN A LA COMPARACIÓN DE IMAGENES

En la última parte del trabajo, se quiere comparar todas las imágenes obtenidas para finalmente encontrar las que mejor representan el Texto original. Es decir, tenemos N conjuntos de imágenes y queremos encontrar las que están más cercanas a todos los conjuntos.

Dados dos conjuntos A y B con X puntos, sabemos que los puntos más importantes serán los que estén en la intersección de los dos, pero existen ciertos puntos que están cerca de los dos conjuntos, sin llegar a la intersección. Dichos puntos también pueden ser considerados representativos de los dos conjuntos (aunque en menor grado). Los demás puntos que estén muy lejos y cerca únicamente de un conjunto no se consideran buenas representaciones. [9]

Siguiendo esta teoría, lo primero que deseamos es encontrar una forma de medir la distancia entre dos imágenes, es decir, una medida de similitud.

Encontrar una medida que nos dé el grado de similitud entre dos imágenes es una tarea compleja, principalmente porque muchas veces no nos damos cuenta de que al comparar dos imágenes, estamos comparando su contenido, y no sus colores, ejes, etc. [11]. Es decir, podemos tener dos imágenes de un gato en diferentes sitios y las consideraremos similares, pero en realidad estamos hablando del contenido.

El problema está en que el ordenador y los algoritmos no pueden simplemente “ver” la imagen y “compararla” para ver si presentan el mismo contenido. La comparación de contenidos incluye un conjunto de variables que se tienen en cuenta: cada objeto en la imagen tiene un formato, tamaño, orientación, colores (uno o más), textura, etc. Y el mismo objeto puede aparecer en dos imágenes distintas pero en otro ángulo, con otro color,… [11]

5.1.1 - CÁLCULO DE LA DISTANCIA ENTRE DOS IMÁGENES

Teniendo en cuenta lo que se ha comentado antes, buscamos una medida de similitud entre dos imágenes basada en la extracción de determinadas características.

La primera idea explorada fue realizar la comparación basándose en el algoritmo SIFT (*Scale-invariant feature transform*). Dicho algoritmo se utiliza para extraer características distintivas de las imágenes [12], siendo una herramienta muy útil para la detección de objetos.

SIFT funciona detectando los *puntos de interés* de una imagen y los busca en la otra imagen, independiente de su orientación o tamaño [13]. La idea de comparación con este método sería obtener el número de puntos de interés que las dos imágenes tienen en común, intentando

obtener un número mayor para imágenes similares y menor para imágenes muy distintas. Normalmente se puede aplicar un "filtro" a los puntos de interés obtenidos, de manera a eliminar puntos falsos.

Existen diversas implementaciones de SIFT en la red. Tras probar las versiones de *OpenIMAJ* y *OpenCV* (las más conocidas), se ha observado que los resultados no eran ideales, por lo que no se puede considerar como una medida fiable.

En la siguiente tabla se muestran algunos de los resultados obtenidos.

Imagen 1	**Imagen 2**	**Resultado** (puntos de interés presentes en las dos imágenes)	**Resultado** (puntos de interés presentes en las dos imágenes) con filtro
 Figura 5 – Ejemplo SIFT: carwindow1.jpg	 Figura 6 – Ejemplo SIFT:carwindow2.jpg	**962**	**8**
	 Figura 7 – Ejemplo SIFT:dog.jpg	**2043**	**7**
 Figura 8 – Ejemplo SIFT:sky.png	 Figura 9 - Ejemplo SIFT: sky2.jpg	**450**	**4**
		**2043**	**5**

Tabla 7 – Resultados para la comparación de imágenes con SIFT

Como se puede observar en la tabla 4, esta medida no es fiable ya que dos imágenes que se podían considerar parecidas, nos da valores contradictorios. Además, se ha observado que el umbral utilizado por la herramienta no mejora el resultado, sino que introduce más incertidumbre al problema.

La segunda solución buscada fue una librería para comparación de imágenes *JPEG* desarrollada en *Java Image Processing Cookbook* [11], que utiliza técnicas de comparación de colores RGB por áreas de la imagen. Los resultados se indican en la siguiente tabla:

<table>
<tr><th>Imagen 1</th><th>Imagen 2</th><th>Resultado</th></tr>
<tr><td rowspan="2"> 
Figura 10 - Ejemplo Java: carwindow1.jpg</td><td> 
Figura 11 - Ejemplo Java:carwindow2.jpg</td><td>2735.98</td></tr>
<tr><td> 
Figura 12 – Ejemplo Java:dog.jpg</td><td>2714.75</td></tr>
<tr><td rowspan="2"> 
Figura 13 – Ejemplo Java:sky.png</td><td> 
Figura 14 - Ejemplo Java: sky2.jpg</td><td>2518.48</td></tr>
<tr><td>  </td><td>4634.06</td></tr>
</table>

Tabla 8 – Resultados comparación con herramienta JAVA

Por las pruebas realizadas se observa que la herramienta proporciona mejores resultados que la anterior, llegando a ser bastante fiable para ciertos ejemplos probados. Pero, en muchos casos, sigue proporcionando un resultado distinto al esperado (ejemplo: Al comparar la Figura 10, con la Figura 11 y 12, se esperaba un resultado mejor para la 11).

5.1.2 - MÉTODO DE COMPARACIÓN DESARROLLADO

Frente a todas esas pruebas, vemos que para conseguir una medida que sea realmente fiable debemos comparar más características de la imagen. Además, es ideal compararlas por bloques, es decir, comparar dichas características con las mismas áreas en ambas imágenes. La medida de comparación elegida se consigue aplicando un proceso de dos partes.

Cada parte analiza una característica de la imagen, dividiéndola en bloques de NxN (en este trabajo 8x8), permitiendo una comparación de las mismas áreas de cada imagen.

Parte 1: Vector obtenido a través del histograma de color de la imagen

"El histograma de color de una imagen es una representación de la distribución del color en una imagen, representando el número de pixeles de la imagen que tienen colores en cada una de las listas de rangos de colores." [15]

Por cada bloque de la imagen se calculan 3 histogramas, uno por cada color (RGB - Red, Green, Blue), y para cada uno de ellos se calcula su media y desviación típica.

La media del histograma se calcula sumando todos los valores obtenidos y dividiéndolos entre el número total de datos:

$$(\sum_{i,j} img(i,j))/N$$

Ecuación 3 – Cálculo de la media del histograma

La desviación típica se calcula siguiendo la ecuación 4, donde N es el número total de datos:

$$\sqrt{(\sum_{i,j}(img(i,j)-media)^2)/N-1}$$

Ecuación 4 – Cálculo de la desviación típica del histograma

Al finalizar la primera parte se obtiene un vector de NxN (8x8) valores de media y varianza (128 en total) multiplicado por 3, ya que se analizan los tres colores RGB.

Parte 2: Vector obtenido a través de la detección de bordes de la imagen

"La detección de bordes es utilizada en el procesamiento de imágenes para identificar puntos en que el brillo de la imagen tiene discontinuidades" [16]. De esta forma, se busca reducir la cantidad de datos en la imagen manteniendo su estructura que puede ser usada posteriormente en el procesamiento de imágenes. [17]

El algoritmo de detección de bordes más conocido y que se utilizará en este trabajo es el algoritmo de *Canny* [17]. *Canny* se basa en 5 fases para conseguir los resultados deseados: Suavizar la imagen, Obtención de gradientes, No supresión máxima, Umbralización doble y Seguimiento por histéresis. [18]

El primer paso realizado es la aplicación del algoritmo de *Canny* a la imagen que queremos comparar, obteniendo una nueva imagen con los bordes encontrados. Dicha imagen será dividida en NxN bloques (así como en el paso anterior), y para cada uno se obtendrá su histograma y se calculará su media (Ecuación 3) y desviación típica (Ecuación 4). El vector obtenido en esta parte está compuesto por 128 valores (8x8 – media y desviación típica).

Comparar dos imágenes a partir de los vectores obtenidos

Una vez realizados los pasos anteriores para las dos imágenes que se van a comparar, tenemos para cada una un vector de 3 dimensiones (R, G, B) con los valores (media y desviación típica) obtenidos a través del histograma de color, y un vector con la información obtenida a partir de la detección de bordes.

Existen diversas maneras de calcular la distancia entre los vectores para obtener la medida final. Las dos que se han analizado se indican en las ecuaciones 5 y 6. Los resultados varían un poco dependiendo del método utilizado, pero dicha variación siempre sigue el margen de dos imágenes calculadas como peores/mejores que la otra. En este trabajo se ha utilizado la primera opción (Ecuación 5), pero sería muy sencillo cambiar a la otra ecuación si se deseara en el futuro.

La ecuación elegida se aplica para cada uno de los vectores, obteniendo un único valor (media, desviación típica) para cada uno.

$$d_i^2 = \sum_i^n (a_{i-b_i})^2$$

Ecuación 5 – Distancia euclídea entre vectores

$$d_i = \sum_i^n |a_i - b_i|$$

Ecuación 6 – Distancia Manhattan

El valor final para la comparación del histograma (paso 1) será la suma de la media y desviación típica obtenida en los vectores R, G y B, mientras que para la comparación de bordes (paso 2) será la suma de la media y varianza del vector E obtenido.

La medida de similitud final se calcula multiplicando λ por el valor obtenido de la comparación del histograma de color y (1- λ) por el valor de la comparación de los bordes. La constante λ actúa como un peso, permitiendo dar más importancia a una característica o a la otra, de manera que se puedan obtener mejores resultados. En la siguiente imagen se ejemplifica el proceso de comparación descrito.

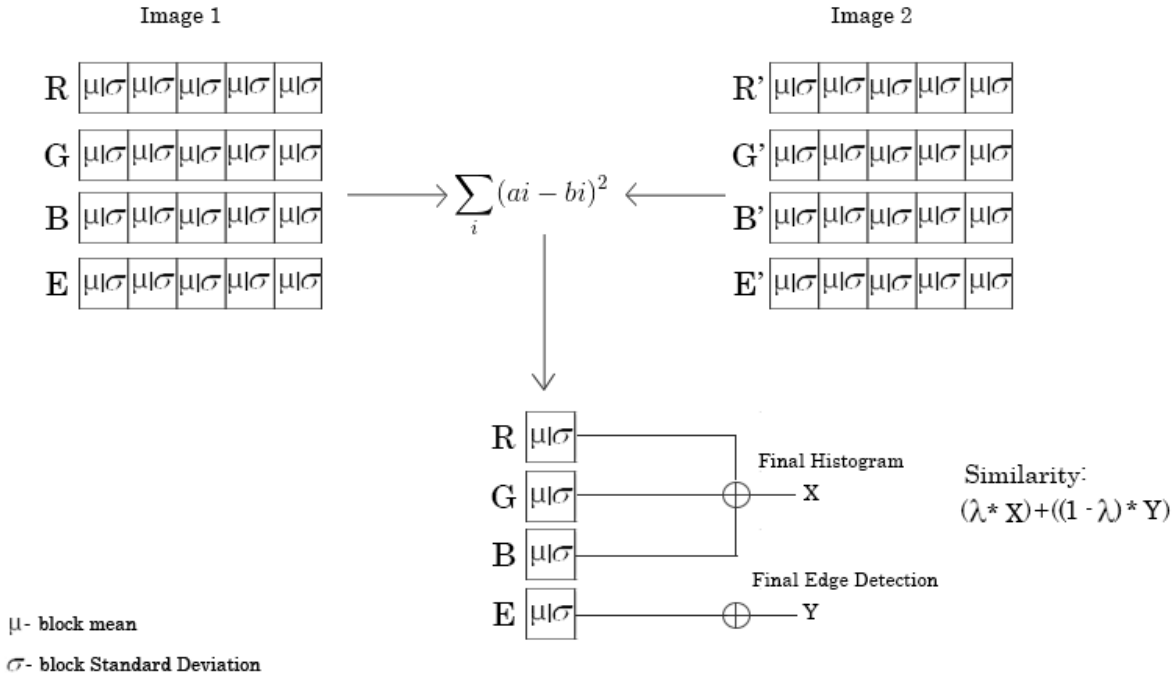


Figura 15 – Diagrama de la comparación de dos imágenes

Intersección de los N segmentos

Tras establecer la medida que se usará para determinar la similitud de dos imágenes, pasamos a comparar todas las imágenes obtenidas para encontrar las que mejor representan a los N segmentos.

Como se ha comentado, no solo queremos las imágenes que se consideran que están en la "intersección" de los conjuntos, sino también las que están muy cercanas. Siguiendo la teoría de [10], se asigna una distancia d_i a cada imagen que se obtiene a partir de su comparación con todas las de los demás segmentos, como se explica a continuación:

- Primero se obtiene una única distancia entre la imagen que se está comparando y cada uno de los demás conjuntos de imágenes (segmentos). Esta distancia será la similitud **MÍNIMA** encontrada al comparar la imagen con todas las que pertenecen al otro conjunto (segmento).
- Una vez que se obtienen N distancias, cada una correspondiente a la distancia entre la imagen y los demás conjuntos, se selecciona el **MÁXIMO**. De esta forma, se logra encontrar el peso final de esta imagen.

Sean A (imgA1, imgA2), B (imgB1, imgB2), C (imgC1, imgC2) tres segmentos con dos imágenes asociadas a cada uno:

- Sea **X** el mínimo obtenido tras comparar “imgA1” con “imgB1” y “imgB2”. E **Y** el mínimo obtenido tras comparar “imgA1” con “imgC1” y “imgC2”.
- El peso de la imagen “imgA1” será el max(X, Y).
- Siguiendo este razonamiento, se calculan los demás pesos.

En la siguiente imagen se muestra un ejemplo de este algoritmo en funcionamiento.

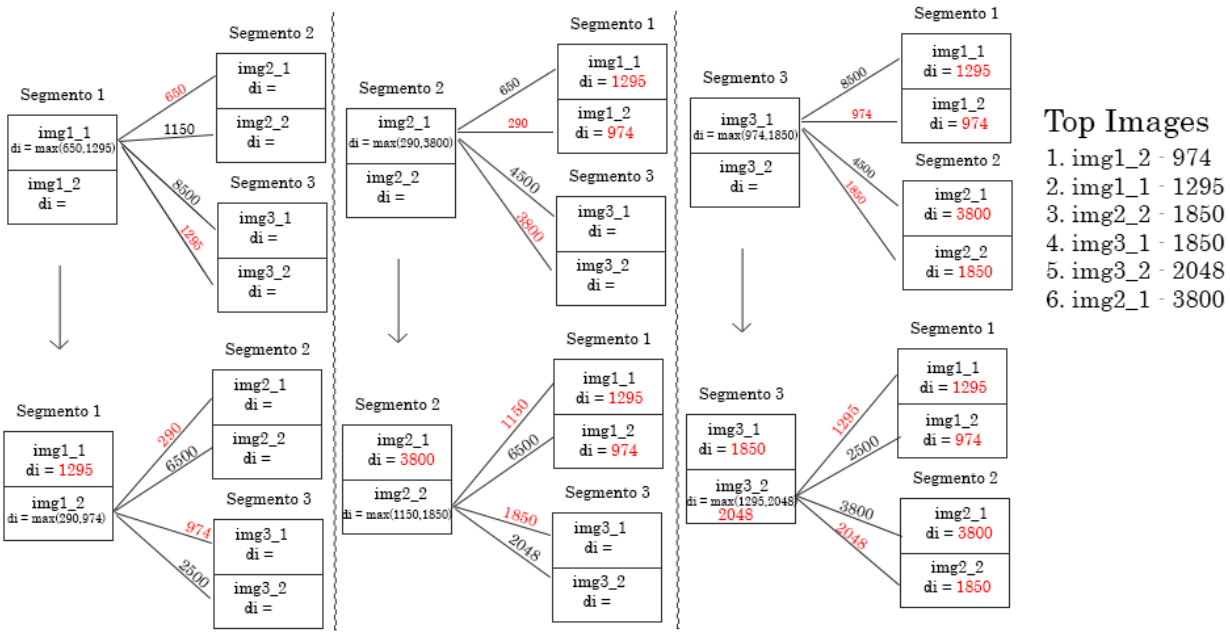


Figura 16 – Ejemplo del algoritmo de comparación

5.2 DESARROLLO

Para esta parte, se ha desarrollado una librería en JAVA que dado el número de segmentos (conjunto de imágenes) existentes en la ruta del programa principal, devuelve la lista con las mejores imágenes ordenada de mejor → peor. Los ficheros que la componen son:

- ImageComparison.java: Clase JAVA que contiene los métodos implementados para la comparación de dos imágenes, siguiendo el proceso de 2 partes comentado al inicio de este apartado.
- BestImagesFinder.java: Clase JAVA que contiene los métodos implementados para obtener las imágenes de los directorios de cada segmento, calcular sus pesos y devolver las mejores.

Las siguientes constantes vienen con un valor ya definido en el programa, pero se puede modificar futuramente en el código:

- *nBlocks*: número de bloques por imagen. Por defecto: 8.
- *K*: Corresponde a la constante λ de la teoría. Por defecto está definida a 0.85, ya que tras la realización de pruebas, fue el valor que proporcionó los mejores resultados.
- *lowerThreshold*: Umbral inferior utilizado en el algoritmo de *CANNY* para la detección de bordes. Por defecto: 50.
- *upperThreshold*: Umbral superior utilizado en el algoritmo de *CANNY* para la detección de bordes. Por defecto: 200.

5.2.1 - OPENCV

Trabajar con imágenes utilizando las librerías ya definidas en JAVA es una tarea compleja. Los métodos son bastante limitados en cuanto a las operaciones que permite realizar con las imágenes. Sin embargo, existe una serie de librerías especializadas en el procesamiento de imágenes que se han desarrollado con el fin de mejorar esta tarea y permitir realizar operaciones más complejas.

Dos de las más conocidas son *OpenIMAJ* [20] y *OpenCV* [19]. Tras realizar pruebas con las dos y analizar los servicios que proporcionan, se ha decidido usar *OpenCV* por tener más documentación disponible y tutoriales para llevar a cabo estas tareas.

OpenCV es una librería open-source de visión artificial bastante útil para el procesamiento de imágenes, que incluye cientos de algoritmos que facilitan su manipulación. Originalmente desarrollada para C++, se ha extendido a otros lenguajes como C, Python y JAVA.

5.2.2 - IMAGECOMPARISON.JAVA

La clase *ImageComparison* desarrollada, permite comparar dos imágenes utilizando los métodos comentados anteriormente y haciendo uso de los servicios de OpenCV. Las imágenes en OpenCV se almacenan en objetos de tipo *Mat*.

Se han creado dos clases auxiliares para facilitar el almacenamiento de los datos:

- Clase **Data**: Almacena la media y Desviación típica de una imagen.
- Clase **ImageData**: Almacena los vectores con los datos obtenidos en los procesos de comparación.
    - Mat imgData: Guarda la imagen original redimensionada al tamaño común establecido.

- Data [][] red, Green, blue, edges: Vectores de tamaño NxN (8x8) donde se guardan los datos obtenidos para cada bloque de la imagen.
- Int nblocks: Número de bloques en que se ha dividido la imagen.
- Size newSize: Tamaño al que se va a redimensionar todas las imágenes.

Antes de empezar la comparación, las dos imágenes son redimensionadas a una medida común establecida (en este caso 512 * 512) para mejorar la comparación evitando problemas de normalización.

A continuación, a través del método *SetImageData*, se obtienen los valores de cada bloque para las dos medidas deseadas (histograma de color y detección de bordes). El algoritmo seguido es el siguiente:

- Se calcula la anchura y altura de cada bloque (en pixeles), dividiendo el total de pixeles por el número de bloques. En este caso, estos valores serán siempre los mismos, ya que se está redimensionando la imagen.
- Obtiene la información asociada al histograma de color:
  - Divide la imagen en bloques utilizando el método *submat* de OpenCV.
  - Obtiene para cada bloque las componentes de los tres colores (R, G, B), con el método *Core.split* de OpenCV.
  - Calcula el histograma de cada bloque. OpenCV proporciona un método que lo calcula directamente, por lo que lo único que es necesario definir son los datos que se utilizarán: Rango – 0 a 255, y el número de marcos (*bins)* – 20.
  - Se almacena la media y desviación típica de cada uno en los vectores correspondientes.
- Obtiene la información de la detección de bordes:
  - OpenCV incluye una implementación del algoritmo de Canny, por lo que, lo primero que se hace es aplicar dicho algoritmo a toda la imagen.
  - Divide la imagen en bloques y calcula el histograma de cada uno.
  - Almacena la media y desviación típica para cada bloque en el vector *"edges"*.

  En esta parte no hace falta separar las componentes R, G, B, ya que al aplicar el algoritmo de CANNY, la imagen resultante no es en color.

Una vez aplicado el proceso anterior a las dos imágenes que se van a comparar, se llama al método desarrollado *compareImgs*, que devuelve el valor final. El algoritmo seguido es el comentado anteriormente en la Figura 15.

Frente a los valores muy grandes que se estaban obteniendo en la comparación, se divide la distancia entre 1000 antes de devolver el resultado.

5.2.3 - BESTIMAGESFINDER.JAVA

Esta clase es la encargada de llamar a los métodos de comparación de imágenes y recoger las mejores. El algoritmo seguido aquí es el demostrado en la Figura 16, que se ha implementado utilizando colas de prioridad (*PriorityQueue* en JAVA).

Para cada directorio de imágenes obtenido (uno por segmento):

- Obtiene el nombre de las imágenes contenidas en el directorio. JAVA ofrece métodos de manipulación de ficheros, por lo que, para obtener un listado de todas las imágenes contenidas en un directorio solo es necesario llamar a la función *listFiles()*.
- Para cada imagen del directorio analizado:
    - Compara dicha imagen con todos los demás directorios, almacenando siempre la distancia **MÍNIMA** obtenida en una cola de prioridad auxiliar.
    - Asigna a la imagen la distancia **MÁXIMA** obtenida. Para eso, se define la cola auxiliar con el parámetro *Collections.ReverseOrder*, permitiendo almacenar los resultados de mayor a menor.
    - Añade la imagen a la cola de prioridad Final.
- La lista con las mejores imágenes se obtiene recogiendo las imágenes almacenadas en la cola de prioridad, ya en el orden deseado (distancia mínima).

5.3 RESULTADOS

En la siguiente tabla se exponen los datos obtenidos en la comparación de un determinado conjunto de imágenes usando el nuevo método desarrollado.

Imagen 1	Resultados (mejor → peor)			
 Figura 17 – "airplane1.jpg"	 2663.91	 2959.83	 3085.12	 3149.39
	 3538.56	 3722.80	 4914.55	 5672.24
 Figura 18 – "carwindow1.jpg"	 2092.73	 2624.50	 2657.59	 2781.47
	 5072.72	 5672.24	 5835.28	 6797.41
 Figura 19 – "sky.png"	 2663.91	 2700.54	 2796.89	 2874.13
	 3551.68	 3870.97	 5072.72	 5339.77

 Figura 20 – "whiteHouse1.jpg"	 1691.55	 2285.24	 2781.47	 4286.73
	 4914.55	 4981.02	 5339.77	 5988.89

Tabla 9 – Resultados de la comparación de imágenes con el método desarrollado

Analizando la tabla anterior se puede observar que el método desarrollado mejora considerablemente el resultado en comparación a los otros métodos estudiados. Para la mayoría de las imágenes probadas se han obtenido resultados razonables para las 2-3 primeras imágenes que devuelve el programa.

CAPÍTULO 6 - DESARROLLO DEL PROGRAMA PRINCIPAL

Para conectar todos los módulos que componen el proyecto y evitar tener que ejecutarlos de forma independiente, se ha desarrollado un programa en JAVA que se encarga de realizar esta conexión, obteniendo los resultados de cada uno y llevando al siguiente programa. El lenguaje elegido ha sido JAVA ya que la mayoría de los módulos se han desarrollado en dicho lenguaje y presenta más facilidad para ejecutar comandos del sistema de manera interna.

Como se está trabajando con el sistema operativo Windows, es necesario compilar los programas implementados en C de manera que sean compatibles con este sistema operativo. Esto se consigue a través de la herramienta ***lcc***, que ya se ha detallado.

6.1 ESTRUCTURA DEL PROGRAMA

El programa se ha desarrollado usando el entorno de programación *Eclipse* (detallado en el apartado *1.3*), y está compuesto por los siguientes módulos:

- **TextSegmenter**: Incluye las clases desarrolladas en el módulo 1, encargadas de la segmentación del texto (*TextSegmenter.java* y *ServerThread.java*).
- **ImageFinder**: Incluye la clase *ImageFinder.java* que contiene los métodos de búsqueda y descarga de imágenes en la base de datos o a través de la API de Bing.
- **ImageSimilarity**: Incluye las clases del módulo de comparación de imágenes (*BestImagesFinder.java* y *ImageComparison.java*), que comparan las imágenes obtenidas y devuelven el resultado final.
- **Main**: Último modulo desarrollado que realiza la conexión entre los programas desarrollados en C y los módulos en JAVA. Está compuesto por las clases: *Link.java* y *Main.java*.

En la misma ruta en la que se encuentra el programa principal, se incluyen los siguientes directorios:

- Maserver – 1.0.0: Directorio que incluye los ficheros de configuración del servidor de MorphAdorner, necesario para segmentar el texto. El servidor se inicializa ejecutando el archivo “runmaserver.bat”.
- imageDatabase: Contiene los ficheros generados de la base de datos utilizada en la búsqueda de imágenes (caso no se use Bing) y el ejecutable del programa desarrollado (*imageClassification.exe*) para volver a generar el fichero de salida. * Este programa no se

ejecuta desde el programa principal, ya que solo es necesario generar el fichero de salida una vez (*imageDatabase.txt*). Además, produciría un retraso en el tiempo de ejecución, ya que al tratarse de 1 millón de imágenes el programa tarda en media 7 minutos.

- textAnalysis: Directorio donde se encuentra el ejecutable del programa desarrollado en C (*textSegmenter.exe*) y los ficheros necesarios para su ejecución (*Corpus – shortcorpus.txt* y lista de *stopwords – longlist.txt*). Además, los ficheros auxiliares generados en la parte de segmentación y análisis se almacenan en ese directorio.

Clase *Link.java*

Esta clase es la que gestiona las llamadas a los módulos y notifica los mensajes de error que se hayan podido producir. Aquí se definen las constantes que indican los nombres de los ficheros y directorios utilizados, por lo que cualquier cambio de ruta se puede indicar fácilmente en el código:

- stopWordsFileName: "longlist.txt". Nombre del fichero de texto que contiene la lista de stopwords.
- textAnalysisDirectory: "textAnalysis/": Nombre del directorio donde se encuentra el programa en C que analizará los segmentos.
- imageDatabasePath: "imageDatabase/imageDatabase.txt": Indica la ruta a la base de datos que se utiliza para la búsqueda de imágenes (si no se usa bing).

El método principal *work(String FileName)*, recibe el nombre del fichero donde se encuentra el texto original que queremos buscar las imágenes asociadas. Lo primero que realiza es cargar la librería necesaria para el funcionamiento de *OpenCV* (*System.loadLibrary(Core.NATIVE_LIBRARY_NAME)*).

A continuación llama al módulo de segmentación del texto (*TextSegmenter*), pasando por parámetro el nombre del fichero de texto y el nombre del directorio donde se encuentra el programa en C que futuramente analizará los segmentos. El módulo se encarga de la segmentación del texto indicado almacenando cada segmento encontrado en un fichero de texto, y devuelve el número de segmentos encontrados, o 0 si ha habido un error.

El número de segmentos se pasa por parámetro al módulo de Análisis, que analiza cada uno de ellos generando un fichero con la lista de palabras y sus pesos. Para ello, se llama a un método (callTextAnalyser) que abre un nuevo proceso y ejecuta el programa en *C* desarrollado.

Una vez que se han generado todos los ficheros, se llama al módulo de búsqueda de imágenes (*ImageFinder*) pasando la ruta al fichero generado de la base de datos y una variable de tipo *boolean* que indica si la búsqueda se realiza con Bing o la base de datos indicada (Por defecto, se

realizará con Bing). Al finalizar su ejecución, ya se habrán descargado las imágenes encontradas para cada segmento en un directorio independiente.

Por último, se llama al módulo de comparación que calcula la similitud de todas las imágenes y devuelve una lista ordenada con los mejores resultados.

La clase **Main.java** define el método *Main* que comprueba si los parámetros pasados al programa son correctos, llama a la clase *link* y finaliza el servidor de MorphAdorner una vez que se haya terminado. En el siguiente apartado se incluyen los resultados finales para las pruebas realizadas.

CAPÍTULO 7 - PRUEBAS FINALES Y RESULTADOS

A lo largo de este trabajo se han ido incluyendo los resultados obtenidos para las pruebas individuales de cada módulo. Este apartado se centra en el análisis de los resultados obtenidos para el programa principal, es decir, dado un texto de entrada analizamos las imágenes devueltas que se consideran las *mejores* para su representación. El texto de entrada es el mismo usado en las pruebas anteriores (**Anexo A**).

En la *figura 21* se puede visualizar el programa en funcionamiento. Se informa al usuario cada paso que se está realizando, y los posibles errores que se hayan producido.

```
Segmenting original Text...
2014-05-20 18:26:57,684 INFO  - MorphAdorner server initialization finished.
4 segments found.
Analysing segment 1...
Analysing segment 2...
Analysing segment 3...
Analysing segment 4...
Finding images for each segment...
finding images for segment 1
Could not download image: http://s1.reutersmedia.net/resources/r/?m=02&amp;d=20140512&amp;t=2&amp;i=896138259&amp;w=450&amp;fh=&amp;fw=&amp;l
finding images for segment 2
Could not download image: http://blogs.reuters.com/nicholas-wapshott/files/2014/03/putin70.jpg
Could not download image: http://www.opednews.com/populum/uploadphotos/s_300_i_ytimg_com_33_hqdefault_424.gif
finding images for segment 3
finding images for segment 4
Could not download image: http://news.bbcimg.co.uk/media/images/64485000/gif/_64485791_damascus_gouta_624.gif
comparing images...
matching 20 images in segment 1.
matching 19 images in segment 2.
matching 19 images in segment 3.
matching 20 images in segment 4.
Top final images:
image: segment4_16.jpg. Sim: 1023.6713665044238
image: segment1_11.jpg. Sim: 1094.6572320222142
image: segment3_14.jpg. Sim: 1145.6224222498302
image: segment4_3.jpg. Sim: 1145.6224222498302
image: segment4_8.jpg. Sim: 1204.2996063645865
image: segment1_7.jpg. Sim: 1244.43310272469
image: segment1_10.jpg. Sim: 1306.2945652585845
image: segment4_13.jpg. Sim: 1345.5119345876008
image: segment1_2.jpg. Sim: 1379.4110984323165
```

Figura 21 – Programa final en funcionamiento

La *tabla 10* incluye las 10 mejores imágenes devueltas por el programa con su peso. Cuanto menor sea el peso, mejor la imagen para el texto leído. Entre las 10 mejores, 4 se refieren al tema de Siria, 4 sobre Ucrania y 2 sobre el avión de Malaysia Airlines.

Analizando las 20 mejores imágenes se obtiene que un 55% viene del segmento de Ucrania, un 30% de Siria y un 15% de Malaysia Airlines. Este último presenta un valor menor que los dos primeros debido al grado de similitud entre los temas de Ucrania y Siria y, en consecuencia, de las imágenes obtenidas para ellos. En el momento de realizar la comparación, las imágenes de estos temas tendrán valores mejores que las de Malaysia.

Aun así, estos resultados pueden ser considerados buenos ya que incluyen imágenes de todos los temas tratados en el texto y la calidad obtenida en cada imagen supone una buena representación.

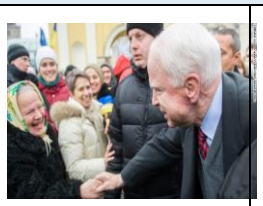
Resultados (mejor → peor)				
				
1023.67 Siria	**1094.65** Ucrania	**1145.62** Malaysia Airlines	**1145.62** Siria	**1204.29** Siria
				
1244.43 Ucrania	**1306.29** Ucrania	**1345.51** Siria	**1379.41** Ucrania	**1393.44** Malaysia Airlines

Tabla 10 – Resultados Finales

Si cambiamos el método de cálculo de distancias entre vectores para la distancia Manhattan (*ecuación 6*), se observa (*tabla 11*) que los resultados no son muy distintos, manteniendo más o menos el mismo porcentaje anterior. Se ha mantenido en el programa final la utilización de la distancia euclídea en vez de la distancia Manhattan ya que en la primera se han obtenido más resultados del segundo tema, mejorando la variedad de imágenes.

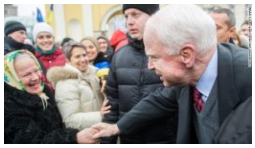
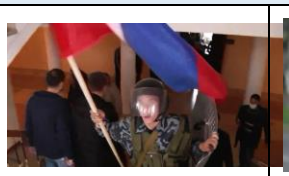
Resultados (mejor → peor)				
				
Malaysia Airlines	Siria	Ucrania	Ucrania	Siria
				
Siria	Ucrania	Ucrania	Ucrania	Siria

Tabla 11 – Resultados Finales (utilizando la distancia Manhattan)

Es importante resaltar que existe una compensación (*trade-off*) entre la calidad del resultado y el tiempo/recursos utilizados por el programa. En las pruebas destacadas en este trabajo se ha descargado siempre un máximo de 20 imágenes por segmento, ya que se ha considerado que tanto el tiempo de ejecución del programa como la calidad del resultado eran aceptables.

Si se aumentara el número de imágenes por segmento (50-60), seguramente se obtendrían mejores resultados, ya que las comparaciones se harían con más muestras. Sin embargo, el tiempo de ejecución del programa también crecería bastante, pues incluye la descarga y comparación de imágenes adicionales. Por ello, se considera que los resultados pueden ajustarse según la necesidad del usuario (tiempo de respuesta más bajo → peores resultados, tiempo de respuesta más alto → mejores resultados).

También cabe destacar que si se aumentara mucho el número de imágenes, la calidad de los resultados volvería a caer, debido a una mayor mezcla de imágenes que no son tan específicas para el tema.

CAPÍTULO 8 - CONCLUSIONES

En este trabajo de fin de grado se han investigado métodos que faciliten la búsqueda de imágenes para documentos cuyo contenido trata distintos temas. Esta tarea incluye el desarrollo de diversos módulos que interactúan unos con otros para obtener los resultados deseados.

En primer lugar, el texto debe ser segmentado y cada segmento obtenido será analizado, con el fin de obtener las palabras clave que componen cada tema. Con esta información se realiza una búsqueda (en motores de búsqueda o bases de datos) descargando una serie de imágenes. Para finalizar, se comparan todas las imágenes para encontrar las que *mejor* lo representan.

Con este objetivo, se han desarrollado los módulos anteriores usando diversas tecnologías ya existentes como librerías de procesamiento de imágenes, API de motores de búsqueda, etc., y un programa principal que se encarga de conectarlos.

Los resultados obtenidos al utilizar la API de BING han sido adecuados, ya que proporciona imágenes recientes y relevantes a los temas buscados. Además, el algoritmo de comparación aquí implementado también consigue buenos resultados para la mayoría de los casos, por lo que las imágenes finales elegidas por el programa abarcan todos los temas tratados en el texto inicial.

Por último, se ha concluido que hay una compensación (*trade-off*) entre el tiempo de respuesta del programa y la calidad de los resultados obtenidos, de manera que al aumentar el número de imágenes por segmento, se obtienen mejores resultados pero peor tiempo de respuesta.

TRABAJO FUTURO

El trabajo se ha desarrollado en módulos de manera que cualquier futuro cambio o mejora de alguno, se pueda llevar a cabo rápidamente y sin afectar a los demás. Las herramientas y métodos de segmentación, análisis y comparación utilizados son solamente algunos entre una gran cantidad disponible actualmente. Los utilizados en este trabajo se han elegido por presentar las características que buscábamos, por lo que sería muy sencillo cambiar cualquiera de los métodos sustituyendo el módulo correspondiente.

Además, la aplicación final desarrollada solo ofrece la ruta de las imágenes seleccionadas tras la comparación, ya que este era el objetivo en este proyecto. Futuramente, se podría integrar una interfaz gráfica que ofreciera la interacción con el usuario, permitiendo realizar los cambios de métodos dinámicamente y mostrando las imágenes elegidas tras la comparación para su selección.

REFERENCIA

[1] N. Stokes,J. Carthy a and A.F. Smeaton - SeLeCT: A Lexical Cohesion Based News Story Segmentation System, AI Communications, Jan. 2004

[2] T. Brants, F. Chen and I. Tsochantaridis - Topic-Based Document Segmentation with Probabilistic Latent Semantic Analysis.

[3] V. Stoyanov and C. Cardie - Topic Identification for Fine-Grained Opinion Analysisis

[4] Freddy Y. Y. Choi - Advances in domain independent linear text segmentation

[5] C. D. Manning, P. Raghavan and H. Schütze - An Introduction to Information Retrieval [book], 2009 Cambridge.

[6] A. Singhal - Modern Information Retrieval: A Brief overview

[7] BING Search API - https://datamarket.azure.com/dataset/bing/search

[8] BING API Schema Guide - https://onedrive.live.com/view.aspx?resid=9C9479871FBFA822!109&app=Word&authkey=!ACvyZ_MNtngQyCU

[9] Java URLConnection – Class Documentation http://docs.oracle.com/javase/7/docs/api/java/net/URLConnection.html

[10] Simone Santini - Finding Representatives of Families of Sets of points in a Metric Space

[11] How to compare Images - http://www.lac.inpe.br/JIPCookbook/6050-howto-compareimages.jsp

[12] Reconocimiento de Imágenes mediante Scale Invariant Feature Transformation (SIFT) - http://www.frsf.utn.edu.ar/cneisi2010/archivos/04-Reconocimiento_de_Imgenes_SIFT.pdf

[13] SIFT Wikipedia - http://en.wikipedia.org/wiki/Scale-invariant_feature_transform

[14] J. Hare, S. Samangooei and D. Dupplaw - The OpenIMAJ Tutorial

[15] Histograma de color, Wikipedia - http://es.wikipedia.org/wiki/Histograma_de_color

[16] Detección de bordes, Wikipedia - http://es.wikipedia.org/wiki/Detector_de_bordes

[17] Canny Edge Detection - http://www.cse.iitd.ernet.in/~pkalra/csl783/canny.pdf

[18] Algoritmo de Canny - https://sites.google.com/site/graficacion22012/ubaldino/algoritmo-de-canny

[19] OpenCV - http://opencv.org/

[20] OpenIMAJ - http://www.openimaj.org/

[21] About Eclipse - http://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Fint_eclipse.htm

ANEXO A – TEXTO DE PRUEBA

A continuación se dispone el texto utilizado en las pruebas de este trabajo obtenido a partir de tres artículos de periódicos americanos (*USA Today* y *RawStory*"). Las palabras del texto no han sido modificadas y se han copiado exactamente como aparecían en los artículos, pero se han eliminado ciertos fragmentos no relevantes para evitar que el texto quedara muy extenso.

"What's next in eastern Ukraine? [9]

Votes held in eastern Ukraine Sunday to secede from Ukraine could lead to three different possibilities for the country:

Chaos increases — There may be a greater chance of a breakdown of central authority and more reasons for Moscow to question the legitimacy of the central government, says Andrew Weiss, an expert on Russia and Ukraine in the White House under Bill Clinton's administration and a vice president at the Carnegie Endowment for International Peace.

Pro-Ukrainian operations against separatists resulted in more than 40 people killed in street fighting and a building fire in Odessa, and at least 20 separatists killed in Mariupol.

"This will not gain friends (in the east) for the Kiev government and will build pressure for Russian intervention," Weiss said.

Valery Bolotov, center, the self-proclaimed governer of Luhansk, delivers a speech as pro-Russia activists celebrate an independence referendum victory May 12 in Luhansk, Ukraine.

Presidential elections are disrupted or rejected in the east — Separatists are likely to disrupt or block general elections May 25 meant to show Ukraine is largely unified behind a central government in Kiev, said Janusz Bugajski, a senior fellow at the Center for European Policy Analysis, a Washington think-tank.

Separatists "do not consider the government in Kiev legitimate, so they will not consider elections legitimate," Bugajski said. "They will probably try to disrupt polling stations from being set up, threaten election commission members, block roads and monitors. ⋯ They will start violence against them."

If Ukrainian authorities attempt to use force to implement elections, that could result in widespread fighting and create an opportunity for Russia to intervene by sending in forces as peacekeepers. Russia sent such peacekeepers to Moldova's Transnistria region in 1992, when pro-Russian forces there tangled with a West-leaning central government. Russian troops remain there.

Even if elections do take place in the east, separatists are not likely to accept the results, and the region could wind up like a Transnistria enclave, says Professor Angela Stent of Georgetown University.

Such an outcome "would make it difficult for Ukraine to function as a state," Stent said.

[9] What's next in eastern Ukraine? (USA Today) - http://www.usatoday.com/story/news/world/2014/05/12/whats-next-in-east-ukraine/9007113/

People mob polling stations during a disputed referendum in the eastern Ukrainian city of Donetsk. Pro-Russia rebels conducted the referendum on whether to seek autonomy for eastern Ukraine.

Talks go ahead on autonomy in eastern Ukraine — All of Ukraine's presidential candidates running for elections have said they support greater autonomy for Ukraine's regions, including in the pro-Russian east, and all sides back talks for resolving conflict.

The problem, Bugajski says, is the goal of the separatists and Russia for such talks is much different than the limited local governance that would be acceptable to the government in Kiev or to the West.

Russia seeks an arrangement like in Moldova, "where you have a virtual separate state that neutralizes decisions of the central government," Bugajski said. Such an autonomous state would have "a veto on Ukrainian moves to establish closer relations with the European Union" and would come under de facto control of Moscow.

Putin is looking for "the maximum level of interference that will produce the minimum cost to Russia," said Damon Wilson, a former director of European Affairs in the White House under President George W. Bush and an adviser to the Obama administration.

Malaysia military says plane turned back [10]

Malaysian officials backed away today from assertions that the missing Malaysian Airlines flight made it to the Strait of Malacca after turning away from its intended course.

The country's air force chief said in a statement today the missing Boeing 777 may have attempted to turn back before it vanished from radar, but there is no evidence it reached the Strait of Malacca off the western coast of Malaysia.

Gen. Rodzali Daud denied remarks reported by a Malaysian newspaper that he had asserted otherwise, based on military radar tracking.

Meantime, the country's civilian aviation chief, Azharuddin Abdul Rahman, said he could neither confirm nor deny the military's earlier reported remarks, that military radar had tracked the plane as it turned direction and flew in a western direction after ending active transponder transmissions.

The developments contributed to what appeared to be a state of confusion at the highest levels of that country over where the plane might be and deepened the mystery of what happened to flight MH370. There were 239 people aboard, including a crew of 12, on a flight to Beijing.

"There is a possibility of an air turn back. We are still investigating and looking at the radar readings," Rahman said today. It is possible that the radar readings are not definitive, especially if the plane was malfunctioning.

The airliner last transmitted a signal to civilian aviation authorities over the Gulf of Thailand, east of Malaysia and south of Vietnam, about 1:30 a.m., or roughly an hour after it took off from Kuala Lumpur.

[10] Malaysia military says plane turned back (USA Today) - http://www.usatoday.com/story/news/world/2014/03/11/malaysia-airlines-investigation/6282557/

U.N. report concludes that chemical weapons were used in Syrian conflict [11]

Chemical weapons have been used at least five times during the Syrian conflict and in some cases children have been slaughtered, according to a UN report released Thursday.

The report cites "credible evidence" and "evidence consistent with the probable use of chemical weapons" in the Syrian sites of Ghouta, Khan Al Asal, Jobar, Saraqueb and Ashrafieh Sahnaya.

But the UN said it could not corroborate their use in two of seven sites studied — Bahhariyeh and Sheik Maqsood.

The United Nations Mission concludes that chemical weapons have been used in the ongoing conflict between the parties in the Syrian Arab Republic," said the report, prepared by a team of experts led by Swede Ake Sellstrom.

However, the report does not attribute blame for the attacks, as this was beyond the mandate given the team by the UN Security Council.

Syrian President Bashar al-Assad has admitted his forces hold chemical weapons, and has vowed to surrender them to international experts. But he insists his forces did not target civilians.

Western and Arab governments, human rights groups and Syrian rebels accuse the regime of carrying out the attacks. Assad and his allies in Moscow and Tehran blame the rebels.

Sellstrom, who led an investigative mission to Syria, had already provided a preliminary report to UN Secretary General Ban Ki-moon on September 16.

That report concluded that banned chemical weapons had been used on a wide scale and that there was clear evidence that sarin gas was used in an attack in the Eastern Ghouta neighborhood near Damascus on August 21.

The final report said the mission "collected clear and convincing evidence that chemical weapons were used also against civilians, including children, on a relatively large scale" on that day in Ghouta.

[11] U.N. report concludes that chemical weapons were used in Syrian conflict (Raw Story) - http://www.rawstory.com/rs/2013/12/12/u-n-report-concludes-that-chemical-weapons-were-used-in-syrian-conflict/

ANEXO B – INSTALACIÓN DEL ENTORNO DE DESARROLLO

A continuación se detalla el proceso de instalación y configuración de las herramientas utilizadas en el desarrollo del proyecto.

ECLIPSE

Eclipse es una plataforma libre de desarrollo que se puede descargar directamente desde su página web[12]. La versión utilizada aquí es "*Eclipse Standard 4.3.2*". La descarga consiste en un archivo *.zip* que contiene carpetas y ficheros de configuración y un ejecutable.

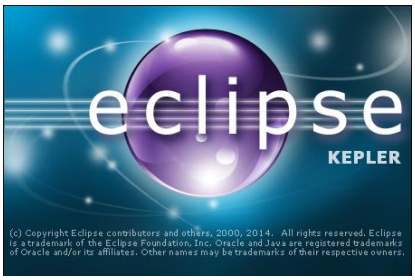


Figura 22 – Inicialización Eclipse

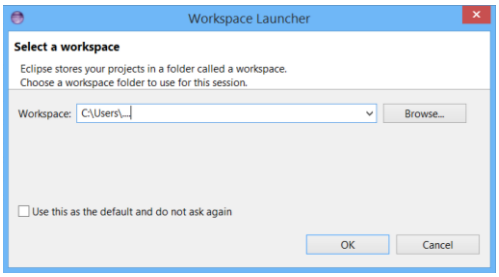


Figura 23 – Selección del espacio de trabajo

Tras ejecutar el programa y definir el espacio de trabajo, el programa ya está listo para ser utilizado. Explicaciones acerca de la utilización de la plataforma y la creación un proyecto se pueden obtener a través de la documentación online[13].

LCC-WIN: COMPILER FOR WINDOWS

El compilador ***LCC*** para el lenguaje de programación C en Windows puede ser descargado desde su página web[14]. Dicho programa es gratis únicamente para uso no comercial, por lo que se puede utilizar en este trabajo.

Tras acceder a la página, se descarga el archivo lcc-win32 o lcc-win64 según el tipo de sistema operativo que se está utilizando. Utilizamos un sistema operativo de 64 bits, por lo que se descarga la versión *lcc-win64*. Esta consiste en un archivo ejecutable que se encarga de instalar las funcionalidades del programa.

[12] Eclipse download: http://www.eclipse.org/downloads/

[13] Eclipse online documentation: http://help.eclipse.org/kepler/index.jsp

[14] Lcc download: http://www.cs.virginia.edu/~lcc-win32/

Finalizada la instalación, es necesario incluir en la variable de entorno *PATH* de Windows la ruta al directorio donde se ha instalado el programa, para que pueda ser utilizado desde el Símbolo de sistema (Command Prompt) en cualquier directorio.

En Windows 8, las variables de entorno se encuentran accediendo a: *Control Panel → System and Security → System → Advanced System Settings → Environment Variables.*

Allí, se selecciona la variable *PATH* y se incluye al final de la misma la ruta de instalación (En este caso: *"C:\lcc\bin"*)

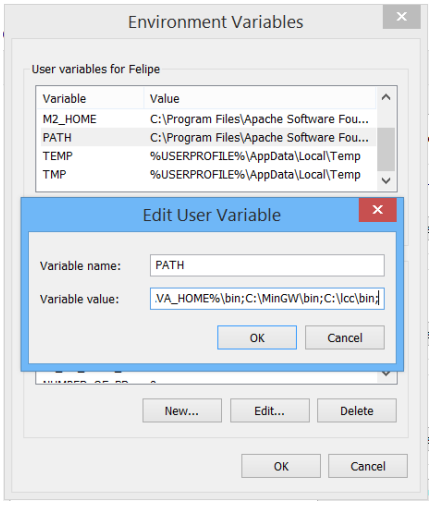


Figura 24 – Editar variable de entorno (Windows)

Ejecución

Una vez configurado, ya se pueden compilar programas en C desde el símbolo de sistema, accediendo al directorio que contiene los ficheros. Es importante resaltar en esta parte que, a diferencia de Linux (compilador gcc), en Windows compilar y enlazar (link) los objetos se hacen de forma separada, por lo que primero es necesario llamar a *lcc* y luego *lcclnk*.

Con el fin de facilitar esta tarea se ha utilizado un *makefile* (Makefile.win), que compila y genera el ejecutable de todos los programas desarrollados en C de una sola vez.

OPENCV

La librería de procesamiento de imágenes *OpenCV* se puede descargar desde su página web[15]. Se selecciona la opción *OpenCV For Windows* según la versión del software deseada (aquí se ha usado la 2.4.8), y una vez descargado el archivo se extrae su contenido.

Este consiste en dos directorios build y sources que deben ser copiados al directorio que el usuario desee (por ejemplo: C:/opencv). Por último, es necesario incluir la ruta de instalación a la variable de entorno *PATH*, como se ha explicado en el apartado anterior.

Configuración en Eclipse

Para poder utilizar los servicios de esta librería en Eclipse es necesario incluir los datos descargados como una *librería de usuario* (*User Library*) en dicho programa. Esto se puede realizar a través del menú *Preferences*, *Java → Build Path → User Libraries → New*, nombrándola OpenCV y luego *Add External JARs* para añadir los ficheros de opencv.

Tras añadir los ficheros, solo es necesario incluir dicha librería en el proyecto creado a través de las propiedades del proyecto.

Más detalles acerca de la instalación y configuración de OpenCV en Eclipse se pueden encontrar en los tutoriales de OpenCV[16][17].

Es importante destacar que antes de poder realizar llamadas a las funciones de *OpenCV* en los proyectos de eclipse, es necesario añadir la línea: ***System.loadLibrary(Core.NATIVE.LIBRARY_NAME)*** al principio del código que se encarga de cargar correctamente la librería.

[15] OpenCV download - http://opencv.org/downloads.html

[16] OpenCV installation: Tutorial - http://docs.opencv.org/doc/tutorials/introduction/windows_install/windows_install.html#windows-install-prebuild

[17] OpenCV configuration for eclipse: Tutorial - http://docs.opencv.org/trunk/doc/tutorials/introduction/java_eclipse/java_eclipse.html